# Managing HP-UX Software With SD-UX

## HP 9000 Computers

HEWLETT PACKARD

HP Part No. B2355-90107
Printed in USA June 1996

E0696

## Notices

## Printing History

```
June 1996 ...  Edition 4.
```

This Edition documents new features applicable to the HP-UX 10.20 operating system. The first edition's Part Number was B2355-90054. The second edition's Part Number was B2355-90080. The third edition's Part Number was B2355-90089.

This guide's printing date and part number indicate its current edition. The printing date changes when a new edition is printed. (Minor corrections and updates which are incorporated at reprint do not cause the date to change.) The part number changes when extensive technical changes are incorporated.

New editions of this manual will incorporate all material updated since the previous edition.

HP Printing Division:

*Open Systems Software Division*
*Hewlett-Packard Co.*
*3404 E. Harmony Rd.*
*Fort Collins, CO 80525*

## About This Manual

This guide provides information on how to install, distribute and manage HP-UX application and operating system software on HP 9000 Series 700 or 800 computer systems using a series of HP-UX commands called SD-UX or Software Distributor-UX.

### Audience

This guide has three audiences:

- System and network administrators who are responsible for installing, distributing, maintaining and administering inventories of software for their system users.
- Software developers or HP Value-Added Business (VAB) vendors who will be packaging their software so it can be used by the SD-UX software management commands.

■ Users of “self-administered,” standalone systems who will also use these utilities to locate, download and update software on their system.

## Conventions

The following typographical conventions are used in this manual:

| | |
|---|---|
| % | A percent sign represents the C shell system prompt. |
| $ | A dollar sign represents the system prompt for the Bourne and Korn shells. |
| # | A number sign represents the Superuser prompt. |
| **%cat** | Boldface type represents command or keywords that you must type. In text, **bold** words are those that are included in the glossary. In examples, text that you enter appears in **bold**. |
| *italic* | Italic text indicates variable values, place holders and function argument names. |
| `Computer text` | Computer typeface or information that the computer displays. Examples of source code and file content listing appear in this typeface. |
| [ ] | Brackets enclose optional items in formats and command descriptions. |
| { } | Braces enclose a list from which you must choose and item in formats and command descriptions. |
| Return or <Return> | Represents a specific key cap on the keyboard. Can be enter, return, shift escape, etc. |
| `Menu Item` | Shaded and boxed computer text signifies a menu choice. |
| \| | A vertical bar separates items in a list of choices. |
| <Ctrl-*x*> | Indicates a control character sequence. Hold down the control key while you depress the key that appears in the place of the *x*. For example, <Ctrl-e> means that you hold down the **Ctrl** key while pressing **e**. |
| Entering Commands | When instructed to *enter* a command, type the command name and then press Return. For example, the instruction “enter the **ls** command” means that you type **ls** and then press Return. |

## Related Documents

In addition to this guide, you can read the following books to give you a more complete view of your HP-UX system and managing software:

- *Using Your HP Workstation* (A2615-90003) Series 700
- *Using HP-UX* (A1700-90014) Series 800
- *Installing HP-UX 10.20 and Updating HP-UX 10.x to 10.20* (B2355-90119)
- *HP-UX Reference* (B2355-90052) 3 Vols.
- *HP-UX System Administration Tasks* (B2355-90079)

## Problem Reporting

If you have any problems with the software or documentation, please contact your local Hewlett-Packard Sales Office or Customer Service Center.

# Contents

## Figures

## Tables

# Part I
# Distributing Software

1

# Introducing SD-UX Software Management

This chapter presents an introduction to HP-UX's software management commands - **swinstall**, **swcopy**, **swremove**, **swlist**, **swconfig**, **swverify**, **swreg**, **swcluster**, **swmodify**, **swpackage**, **swagentd**, **swgettools** and **swacl**—and a short overview of the software management process that uses them. These commands are collectively referred to as SD-UX (for S oftware D istributor-HP -UX ).

*There are important concepts explained in this chapter you must know to understand the SD-UX software management commands and their operation.*

**Note** SD-UX commands manage software on a local host only. To install and manage software simultaneously on multiple remote hosts (including PCs) from a central controller, you must purchase the HP OpenView Software Distributor (HP Prod. No. B1996AA) which provides extended software management and multi-site software distribution capabilities.

SD-UX is based on distributed, client/server technology, and requires some networking functionality on the host system for proper execution. These networking services are only available in Run Level 2 (Multi-User mode) and above. *SD-UX cannot run in Single User mode.*

## SD-UX Commands Overview

All the SD-UX commands listed below can be invoked on the command line. In addition, the `swinstall`, `swcopy` and `swremove` commands offer an interactive Graphical User Interface (**GUI**) with windows and pull-down menus, or a text-based Terminal User Interface (**TUI**) in which screen navigation is done with the keyboard (i.e. using tab and character keys; no mouse). See "Executing SD-UX Commands" in this chapter and Chapter 2 for more information.

The syntax, options, defaults and operands are similar for all these SD-UX commands:

| **Command** | **Description** |
|---|---|
| `swinstall` | Installs or updates software to the local host from a CD-ROM/tape or from a special SD-UX directory called a **depot** (see section "Distribution Depots"). When you use `swinstall`, software is installed directly into the default directory (`/`) or into an alternate directory which you specify. `swinstall` automatically configures your system to run the software when it is installed into the default directory. See Chapter 2 for more information. |
| `swcopy` | Copies software from a CD-ROM/tape into one or more "depots" on the local host. *Software that is copied into a depot cannot be used; it must be installed from that depot to make it usable.* If your system is to act as a software "source" by other systems, you must first *copy* the software from the physical media into a depot. `swcopy` can consolidate many different software products and versions into a depot. Configuration is not done with `swcopy`. See Chapter 2 for more information. |
| `swremove` | Deletes software that has been installed on your system. It also removes software from depots. See Chapter 6 for more information. |
| `swlist` | Displays or lists information about software that is installed on your system, contained in depots or on physical media. See Chapter 5 for more information. |
| `swcluster` | Installs software in an NFS Diskless Cluster by: 1) using `swinstall` to install the software to the cluster server; or 2) using `swinstall -l` to linkinstall the software to the NFS clients, then configuring the software with `swconfig`. See Chapter 2 and Chapter 6 for more information. |

| | |
|---|---|
| `swconfig` | Prepares your system to run software that was installed with `swinstall`. Although configuration is automatically performed as part of the `swinstall` command, `swconfig` explicitly configures, reconfigures or unconfigures a host when these actions are needed separately. See Chapter 3 for more information. |
| `swverify` | Compares the original software files on the source against those that were installed to verify their integrity. Also verifies software that was copied to a depot. See Chapter 3 for more information. |
| `swreg` | Normally, the `swcopy` command automatically **registers** newly created depots to make them "visible" to other systems and the `swremove` command automatically "unregisters" them. `swreg` registers or unregisters depots when these actions are needed separately. See Chapter 4 for more information. |
| `swmodify` | SD-UX commands automatically keep track of software management operations by creating an Installed Products Database (IPD) and various "catalog files" (see section "Installed Products Database" in this chapter for more information) that contain information about the software on the system. Although neither the IPD or catalog files can be edited directly, the `swmodify` command allows you to change the contents of these files via the command line. See Chapter 7 for more information. |
| `swpackage` | Allows software vendors and system administrators to "package" software products onto a tape or depot which is then used as a software source. System administrators can also use this command to repackage existing product filesets for installation. See Chapter 9, Chapter 10 and Chapter 11 for more information. |
| `swagentd` | Software destinations and sources require *daemons* and *agents* to accomplish SD-UX software management tasks. SD-UX commands interact with the daemon (`swagentd`) and agent (`swagent`) running on source and destination systems. The `swagentd` daemon process must be scheduled before a system is available as a destination. The `swagent` process is executed by `swagentd` and is never invoked by the user. |
| `swgettools` | In order to load software products from a new SD media, the local system must first have the SD tools in place that are compatible with the new SD media. This command is used to load these tools from the new media onto systems that do not have updated tools. See Appendix C for more information. |

`swacl` SD-UX software objects can be protected from unauthorized access by **Access Control Lists (ACLs)**. `swacl` lets you specify, view and change these access permissions. See Chapter 8 for more information on SD-UX Software Security.

**Note** All of the above functionality is provided by a software product called `SW-DIST` which is included on your HP-UX 10.X Core OS Disk or Tape. If `SW-DIST` is missing or corrupted on your system, you will NOT be able to install or copy *any* HP-UX software that is in the SD-UX format, *including a new* `SW-DIST` *product*. Refer to Appendix C for information on how to re-load the SD-UX functionality.

## Understanding SD-UX Terms and Concepts

Throughout this guide, the term **host** is used to mean an individual computer, standing alone or connected to a network of other computers. The **local host** is the system on which you are invoking the SD-UX commands. Hosts may also be called **nodes**, **servers**, **clients** or **systems**.

We also refer to **targets** which means the destination host or directory (root or depot) on which the SD-UX operation is to be performed. For most SD-UX operations, your target is your local host or depots that are on it.

A **software source** is a physical medium (CD-ROM or Tape) or directory location (depot) that contains software that is to be installed.

The role of SD-UX **controller** is assumed by the local host on which you invoke the SD-UX commands. Controller programs are the front ends in the SD-UX process, providing the user interface for the management tasks. A controller's role is to collect and validate the data it needs to start a task, and to display information on the task's status.

The SD-UX controller programs communicate with hosts and depots through the SD-UX **agent** called `swagent`. An agent is that part of SD-UX that actually performs the basic software management tasks. The SD-UX daemon that executes the agent is called `swagentd`. On HP-UX 10.X systems, the SD-UX controller and agent are both running on the local host.

## The Roles Systems Play

There are three roles an individual host may play in the SD-UX software management process - Local Host, Distribution Depot (or Server) and Development System. We will refer to these roles throughout this guide. The role a host plays depends on which command is used.

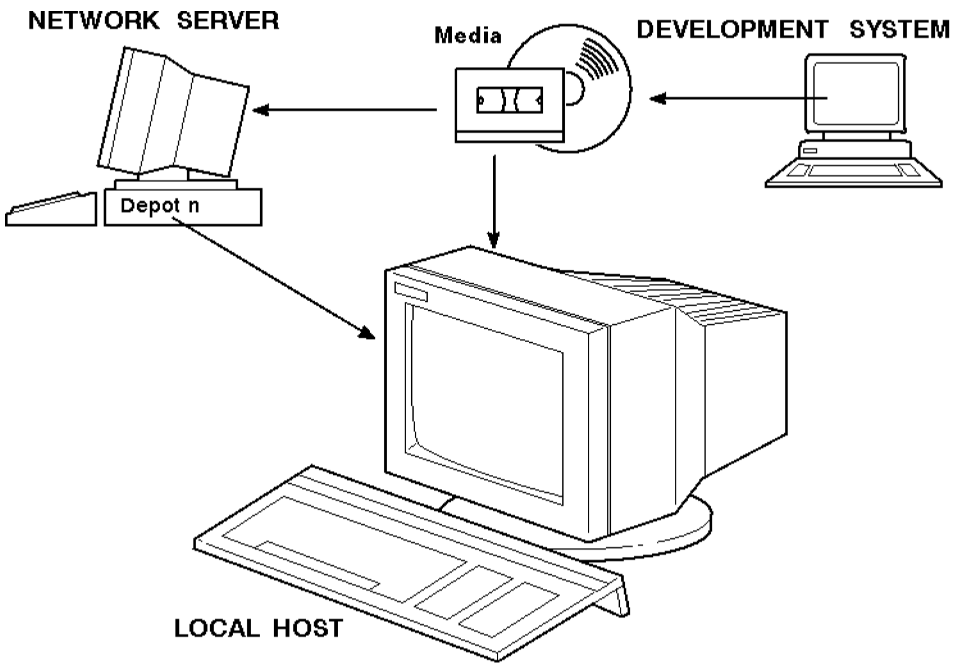


**Figure 1-1. SD-UX Systems**

### Local Host

A local host is any system on which software is to be installed or managed using the SD-UX commands. It is sometimes referred to as the **controller host**.

However, a local host that contains a depot could play a *source* as well as a *destination* or target role when other client systems obtain software from that depot via the network.

### Distribution Depots

A **depot** is a directory location on the local host which is used as a "gathering place" for software products. It is a customizable source of software used for direct installations by the local host or by other hosts on the network.

A depot is created by copying software directly to it from the physical media (using the SD-UX `swcopy` command) or by creating a software package on it (using the `swpackage` command) and then "registering" the depot with `swreg`, see Chapter 9. It can then be used as the source for installation tasks by the `swinstall` command which is executed on the target machine.

There are two types of depots:

**Directory Depot**
: Software in a directory depot is stored under a normal directory on your file system (usually */var/spool/sw*). This software is in a hierarchy of subdirectories and filesets organized according to a specific media format. A directory depot can be "writable" or read-only.

: When using the SD-UX commands, you refer to a directory depot via its top-most directory. In a CD-ROM depot, this directory would be the CD-ROM's "mount point."

**Tape Depot**
: Software in a tape depot is formatted as a `tar` archive. Tape depots such as cartridge tapes, DAT and 9-track tape are referred to by the file system path to the tape drive's device file.

: A tape depot can only be created by using `swpackage` and it cannot be verified or modified with SD-UX software management commands. You cannot copy software (using `swcopy`) directly **to** a tape; use `swpackage` for this operation (see Chapter 9 and Chapter 11 for more information).

: Software in a tape depot must first be transferred to a directory depot before it can be "pulled" by other hosts on the network. A tape depot can be accessed by only one command at a time.

A depot usually exists as a directory location (that is, a directory depot). Therefore, a host may contain several depots. For example, a designated software distribution server on your network might contain a depot of word processing software, a depot of CAD software and a spreadsheet software depot, all on the same server.

**Note** HP Series 700 and Series 800 software cannot coexist in the same depot. If you administer software for both of these platforms, you should create separate depots for each.

### Network Sources

If a depot resides on a system that is connected to a network, then that system can be a **network source** for software. Other systems on the network can install software products from that server instead of installing them each time from a tape or CD-ROM.

Another means of sharing operating systems and file system elements is through an **HP-UX NFS Diskless Cluster**. A diskless cluster consists of a cluster server (which provides a root file system) and one or more cluster clients, all attached to a network. In this way, the cluster server acts as a network source for operating systems and application software.

A network source offers these advantages over installing directly from tape or CD-ROM:

- Several users can "pull" software down to their systems (over the network) without having to transport the tapes or disks to each user.
- Installation from a network server is faster than from tape or CD-ROM.
- Many different software products from multiple tapes, CD-ROMs and network servers can be combined into a single depot serving all others on the network.

### Development Systems

As a software application is developed, files are taken from the programmer's environment and placed on a **developer host** where they are "integrated" for distribution. The SD-UX `swpackage` command prepares these software files by organizing them into specific product, subproduct and fileset structures. It also uses special information files (see Chapter 10) that are used to help other commands identify, distribute and manage the application.

After it is organized, the software on a developer host is then "mastered" or copied onto CD-ROMs or tapes for further distribution to users or customers. The resulting package can also be made network-accessible to users.

## Sharing Software in a Diskless Cluster

SD-UX commands can also install software on a diskless cluster server in a way that makes it available to cluster clients. This software "sharing" is done using two kinds of special directory locations created on the cluster server:

**private roots** A directory on the cluster server that serves as a client's root directory. This directory contains all of the client's private files, directories and mount points for shared files.

HP's System Administration Manager (SAM) creates private roots in the */export/private_roots* directory on the cluster server.

**shared roots** A directory on the cluster server that serves as a source of operating system software and system files for installation in client private root directories.

These shared roots, which must exist on the server before clients can be added to create a cluster, are created in the */export/shared_roots* directory on the cluster server. On Series 700 servers, the server's root directory also serves as a shared root, symbolically linked to */export/shared_roots/OS_700*.

Executables and other normally unchanging directories and files are mounted read-only on corresponding mount points in the client's private root file system. Copies of system configuration files and other modifiable files and directories are copied from the shared root and installed directly in the corresponding locations in the client's private root file system.

For more information on NFS diskless systems, see the "NFS Diskless Concepts and Administration" whitepaper (in PostScript format) located in */usr/share/doc/NFSD_Concepts_Admin.ps* and the chapter "Setting Up and Administering HP-UX NFS Diskless Clusters" in the *HP-UX System Administration Tasks* manual (HP Part No. B2355-90079).

The `swcluster` command lets you share software on an NFS Diskless Cluster. To install (or remove) software on a diskless cluster and then make it available to clients on the cluster, you can use the SD-UX `swcluster` command. This command lets you, with a single operation,

1. perform installations (or removals) on the cluster server,
2. make the software available to the cluster client and
3. configure the software for use on the client system.

SD-UX allows only a single operating system per hardware type in a shared root.

The ''swcluster'' command is a higher level operation than `swinstall -l` (or `swremove -l`) option. When you execute the `swcluster` command, the GUI or TUI for `swinstall` appears on your screen. See "Advanced Topics for `swinstall` and `swcopy`" in Chapter 2 for more information on `swcluster`.

## Installing "Protected" Software

Most HP software products are shipped to you on CD-ROM as "protected" products. That is, they cannot be installed or copied unless a "codeword" and "customer ID" are provided by you. Software that is unlocked by a codeword may only be used on computers for which you have a valid license to use that software. *It is your responsibility to ensure that the codeword and software are used in this manner.* The customer ID uniquely identifies the owner of the codeword.

The *codeword* for a particular software product is found on the CD-ROM certificate you received from HP. It shows the codeword along with the customer ID for which this codeword is valid. One codeword usually unlocks all the products on a CD-ROM which you have purchased. When an additional HP software product is purchased, an additional codeword will be provided by HP. Just enter the new codeword and customer ID and they will be merged with any previously entered codewords.

The *customer_id*, also found on the Software Certificate, lets you restrict installation to a specific owner. SD-UX provides defaults in which you may specify your codeword and customer_id via the command line or the SD-UX Interactive User Interface. See Appendix A and "Using Software Codewords and Customer IDs" in Chapter 2 for more information on codeword and customer_id defaults.

## SD-UX Software Structure

SD-UX commands work on a hierarchy of **software objects** - bundles, products, subproducts and filesets - that make up the applications or operating systems you want to manage.

**Bundles** Collections of filesets, possibly from several different products, that are "encapsulated" by HP for a specific purpose. Bundles can be stored in software depots and copied, installed, removed, listed, configured and verified as single entities. Only HP can create bundles and all HP-UX 10.X OS software is packaged in bundles. Bundles, since they are groups of filesets, are NOT necessarily supersets of products.

**Products** Collections of subproducts (optional) and filesets. The SD-UX commands maintain a product focus but still allow you to specify subproducts and filesets.

Different versions of software can be defined for different platforms and operating systems, as well as different revisions (releases) of the product itself. Several different versions could be included on one distribution media or depot. However, you cannot mix Series 700 and 800 software on a single depot.

**Subproducts** Subproducts are used to group logically related filesets within a product if the product contains several filesets.

**Filesets** Filesets include the all the files and **control scripts** that make up a product. They are the smallest manageable (selectable) SD-UX software object. Filesets can only be part of a single product but they could be included in several different HP-UX bundles.

SD-UX commands refer to this product structure in the form:

```
bundle[.]
```
*or*
```
product[.[subproduct.]fileset]
```

with periods separating each level.

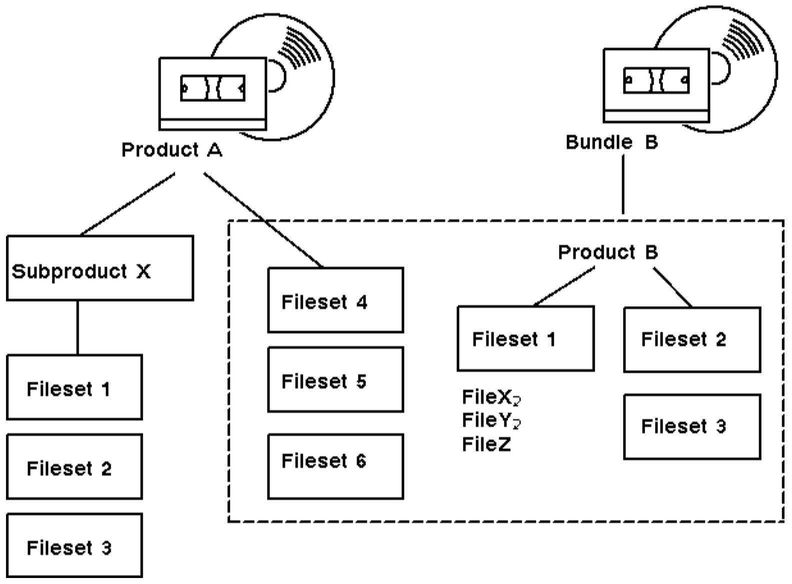
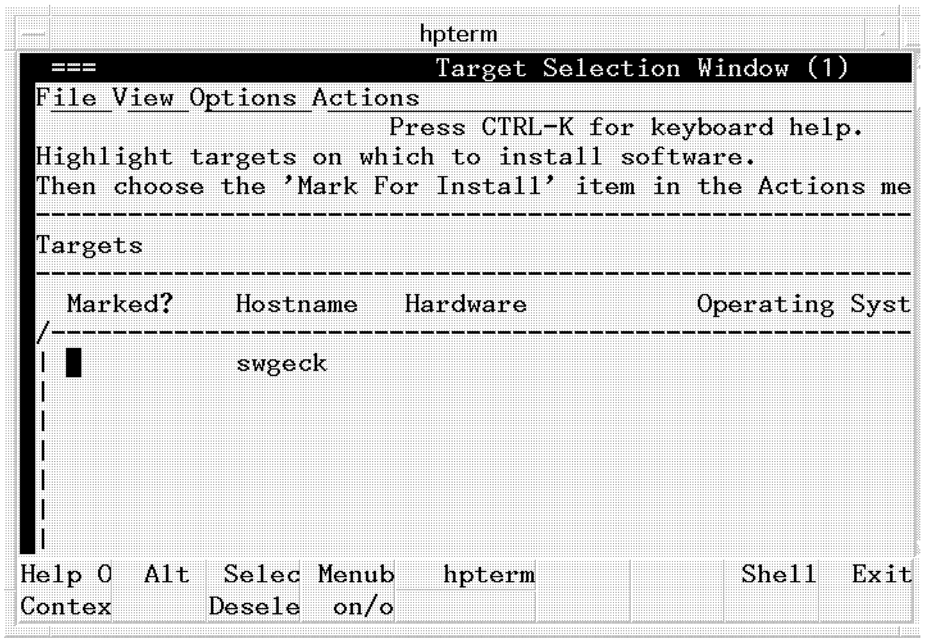


**Figure 1-2. HP-UX Software Structure**

## Executing SD-UX Commands

You can invoke all SD-UX software management commands non-interactively via the command line. The command line interface is used effectively when you want to write command scripts to be executed at a later time or to execute tasks that take a long time to accomplish.

The `swinstall`, `swcopy` and `swremove` commands also provide a Graphical User Interface (GUI). A Terminal User Interface (TUI) for these commands is also supplied for those computers without graphics terminal capabilities. For those management tasks you want to monitor as they are being performed, the GUI (or TUI) is a quick and efficient way to work with the commands, analyze the effects of a task and retry those that might fail.



```
hpterm
 ===                      Target Selection Window (1)
File View Options Actions
                         Press CTRL-K for keyboard help.
Highlight targets on which to install software.
Then choose the 'Mark For Install' item in the Actions me
---------------------------------------------------------
Targets
---------------------------------------------------------
  Marked?    Hostname   Hardware           Operating Syst
/--------------------------------------------------------
|            swgeck
|
|
|
|
|
|
Help O  Alt   Selec Menub   hpterm                Shell   Exit
Contex        Desele on/o
```

**Figure 1-3. The Terminal User Interface (TUI)**

The TUI is the default mode of interacting with `swinstall`, `swcopy` and `swremove` *if you have NOT set your DISPLAY variable.* If you *have* set your DISPLAY variable, invoking these commands will automatically start the GUI. Complete instructions on how to use the GUI are found in Chapter 2.

The GUI or TUI are the primary and suggested methods of interacting with the install, copy and remove operations. The command line interface requires you to be familiar with a broad range of defaults, options and other variables that are available in the SD-UX software management environment.

## Using the Command Line Interface

SD-UX command behaviors, source/host designations and software selections are specified on the command line in one of two ways:

1. By specifying (often multiple) software selections, host names and option values individually on the command line or

2. by using **input files** that contain lists of software selections, variables and operands to use in the command (see the section "Input Files" below for more information).

   Here is what a typical SD-UX command line looks like:

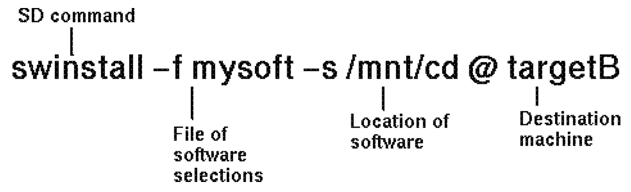

SD command
swinstall –f mysoft –s /mnt/cd @ targetB
File of software selections
Location of software
Destination machine

**Figure 1-4. Sample Command**

**Note** The @ ("at") character *and the required single space following it* are optional, but important, to the successful operation of SD-UX command lines that specify host locations. *They are not necessary if you are specifying the local host and default depot as the recipient system.* If they are used, they act as a separator between operands and the destination. Only one @ character is needed.

On some systems, the @ character is used as the "kill" function. Type `stty` on your system to see if the @ character is mapped to any other function on your system. If it is, remove or change the mapping.

## Input Files

Both the command line and graphical interfaces can be influenced by various **input files**:

**Defaults File** The file */usr/lib/sw/sys.defaults* contains all the SD-UX software management defaults and their values. It is the master "template" file and is not used directly by SD-UX. Individual defaults are copied from this template file, added to your system defaults file stored in */var/adm/sw/defaults* or *$HOME/.sw/defaults* and then their values changed. These values can then be customized, added to, changed or overridden from the command-line.

See Appendix A for a complete listing of defaults and their values and descriptions.

**Session Files** Before any SD-UX task actually starts, the system automatically saves the current command options, source information, software selections and host designation into a **session file** (in the *$HOME/.sw/sessions/* directory) as `<swcommand>.last`. Each time you save a session file, it will overwrite the previously stored one. To save multiple session files, just rename the `<swcommand>.last` file.

Once you have performed an installation, you need not "start from scratch" the next time; just recall a previously stored session file. See "Managing Sessions - The `File` Menu" in Chapter 2 for more information.

**Software Selection Files** The */var/adm/sw/software/* directory is reserved for files containing the names of software selections to be installed, copied or removed. To keep the command line shorter, software selection input files let you specify long lists of software products. You can imagine how long the command would be if you were installing ten different software products and had to specify each one by name on the command line. With a *software selection file*, you only have to specify the single file name.

**Host Files** The *defaults.hosts* file contains lists of hosts that are used by the GUI and TUI to allow pre-selected choices for sources and targets. These lists are stored in the *$HOME/.sw/defaults.hosts* or */var/adm/defaults.hosts* files.

See each command's chapter in this guide for complete descriptions of the options, defaults, input files and operands.

## Using the Graphical or Terminal User Interfaces

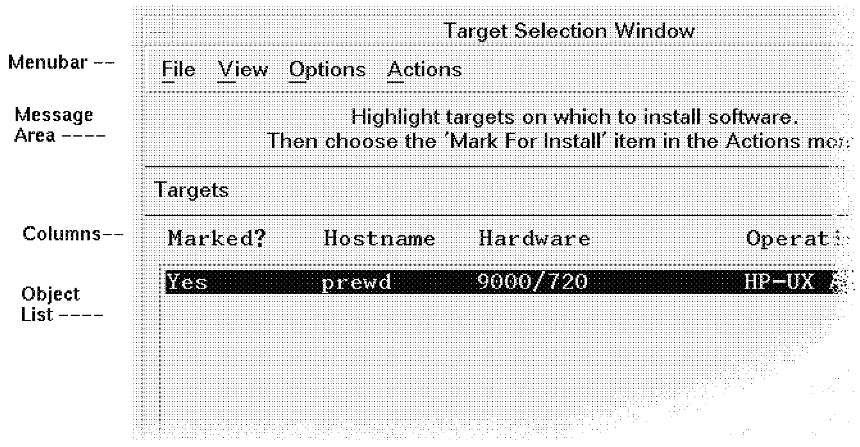
The Graphical User Interface helps you designate sources, specify software products and perform other operations by choosing actions from pull-down menus, filling out dialog boxes and highlighting choices from object lists. It also allows you to visually monitor a particular process and get a better overall picture. The Terminal User Interface also allows interaction with menus and object lists via ASCII character screens. The Graphical User Interface can also be launched from inside HP's Systems Administration Manager (SAM) application.

Only the `swinstall`, `swcopy` and `swremove` commands feature this Graphical or Terminal Interface. The `swlist`, `swverify`, `swconfig`, `swreg`, `swmodify`, `swacl` and `swpackage` commands rely on the command line interface for specification and execution.

To start the Graphical or Terminal User Interface for an install, copy or remove session, type:

```
/usr/sbin/swinstall
   or
/usr/sbin/swcopy
   or
/usr/sbin/swremove
```

If you put `/usr/sbin` in your PATH, you can dispense with the `/usr/sbin` prefix.



Highlight targets on which to install software.
Then choose the 'Mark For Install' item in the Actions me

```
Marked?   Hostname   Hardware       Operat
Yes       prewd      9000/720       HP-UX
```

**Figure 1-5. GUI Window Components**

The major Graphical and Terminal User Interface Windows contain a Menubar across the top, a Message Area, Column Headings and an Object List.

| | |
|---|---|
| Menubar | The Menubar provides menu choices such as: `File View Options Actions ...... Help`. Each of these choices have additional "pull-down" menus for more activities. Placing your mouse cursor on the appropriate Menubar choice and "clicking" (pressing the left mouse button) |

causes additional menus to appear. In the TUI use the `Arrow`, `Tab` and `Return` keys. Items within the menus may appear or disappear depending on whether selections are highlighted or not. Some items may also be "grayed" to show they are not available for a specific action.

Message Area — The Message Area provides information on how to navigate through the various windows and how to select items. The Message area is for your information only; items in it cannot be changed by the user.

Object List — The Object List contains the name of the targets, selections or other information regarding selections, analysis and details. Flags ("Yes," "Partial" or blank) are used to denote whether items in the list have been "marked" for an activity (see the "Marked?" column). Items in the Object List can be marked by highlighting (clicking on them with the left mouse button). If you have a Terminal User Interface, you can mark (choose) objects by pressing `Return` when the focus is on the item and then pressing the "m" key on your keyboard. Unmark items in the TUI by using the "u" key.

The Graphical User Interface also allows you to change window view preferences to fit your requirements. Using a Columns Editor dialog (see Chapter 2), accessible through the `View` menu item, you can customize window appearances by changing Object List column content, width, justification and the position of attributes or headers.

## XToolkit Options and Changing Display Fonts

The SD-UX commands support the following subset of the HP-UX XToolkit command line options:

- `-bg` or `-background`
- `-fg` or `-foreground`
- `-display`
- `-name`
- `-xrm`

Note that the SD-UX software management commands do not support the XToolkit `-fn` or `-font` option used to change display fonts. However, there is an alternative method for controlling your display fonts.

SD-UX commands recognize most Motif™ standard resources when running in the X11/Motif environment, plus the following additional resources:

| | |
|---|---|
| `*systemFont` | Specifies the variable width font used in the GUI menubars and other areas where a variable width font is applicable. The default size is 8x13. |
| `*userFont` | Specifies the fixed width font used in all other GUI displays. This font should be the same basic size as the `*systemFont` only in the fixed width style. The default size is also 8x13. |

Here is an example of how to change the size of your fixed width font from 8x13 to 6x13:

```
swinstall -xrm 'Swinstall*userFont: user6x13'
```

Here is how to change the variable width font style to 12 point HP Roman 8:

```
swinstall -xrm 'Swinstall*systemFont: -adobe-courier-medium-r-normal--12-120-75-75-m-70-hp-roman8'
```

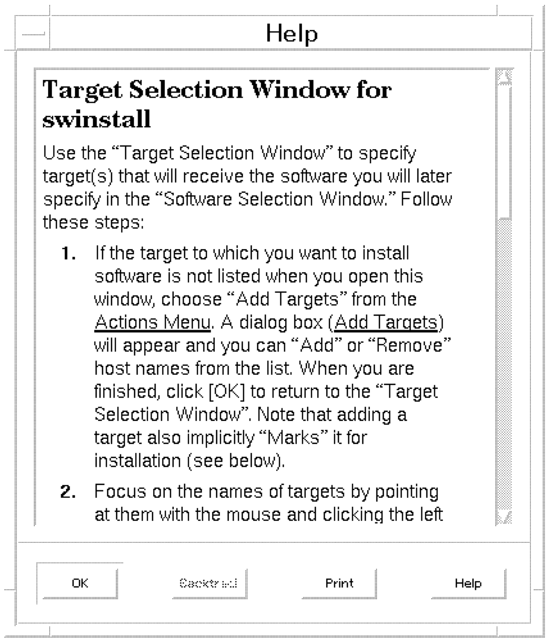
You can also create a defaults file (in */usr/lib/X11/app-defaults*) for each command with a Graphical User Interface so that a resource will be set each time you invoke a specific command. Here is an example of an app-defaults file for `swremove`:

```
# Swremove app-defaults

Swremove*foreground:  red
Swremove*background:  white
Swremove*userFont:    hp8.8x16b
Swremove*systemFont:  -adobe-courier-medium-r-normal--12-120-75-75-m-70-hp-roman8
```

## Using the On-line Help System

The SD-UX `swinstall`, `swcopy` and `swremove` commands have an on-line help system to assist you in working with the GUI or TUI. SD-UX menu choices and activities have associated help screens that explain the activity, provide examples and answer your questions about software management tasks.



**Target Selection Window for swinstall**

Use the "Target Selection Window" to specify target(s) that will receive the software you will later specify in the "Software Selection Window." Follow these steps:

1. If the target to which you want to install software is not listed when you open this window, choose "Add Targets" from the Actions Menu. A dialog box (Add Targets) will appear and you can "Add" or "Remove" host names from the list. When you are finished, click [OK] to return to the "Target Selection Window". Note that adding a target also implicitly "Marks" it for installation (see below).
2. Focus on the names of targets by pointing at them with the mouse and clicking the left

**Figure 1-6. A Typical On-line Help Screen**

For a description of each menu choice in the various screens, windows and menus, place the cursor on an item with the left mouse button or arrow keys and press the `f1` key on your keyboard. This displays a Help Screen that provides additional information on that item. For an *overview* of each major screen, plus help on the keyboard and other product information, choose the `Help` menu item in the Menubar.

For more complete technical information on each of the SD-UX commands, use the HP-UX `man` command (`man <SD-UX command>`) to see the individual SD-UX manual pages.

## Installed Products Database

SD-UX keeps track of software installations, products and filesets on your system with an **Installed Products Database (IPD)** for root installed software and **catalog files** for software in depots.

Located in the directory */var/adm/sw/products*, the IPD is a series of files and subdirectories that contain information about all the products that are installed under the root directory (/). This information includes 14-character "tags" or product names, one-line title fields, paragraph-or-longer description text, long README files, copyright information, vendor information and part numbers on each product installed. In addition, the IPD contains revision information and a user-targeted **architecture** field including the four **uname attributes** - operating system name, release, version and hardware machine type. Here is what the IPD *INFO* file for a product called "Accounting" looks like:

```
fileset
tag ACCOUNTNG
data_model_revision 2.10
instance_id 1
control_directory ACCOUNTNG
size 292271
revision B.10.10.
description Vendor Name: Hewlett-Packard Company
Product Name: Accounting
Fileset Name: ACCOUNTING

Text: "HP-UX System Accounting feature set.  Use these features to
gather billing data for such items as disk space usage, connect time
or CPU resource usage.
"
timestamp 797724879
install_date 199504121614.39
install_source hpfclc.fc.hp.com:/release/s700_10.10_gsL/goodsystem
state configured
ancestor HPUX9.ACCOUNTNG
corequisite OS-Core.CMDS-AUX,r>=B.10.10,a=HP-UX_B.10.10_700,v=HP
```

Catalog files are the equivalent IPD files but they are for software stored in a depot. When a depot is created or modified using `swcopy`, these files are created and placed in the specified depot (or in the default */var/spool/sw* depot). They describe the depot and its contents.

The `swinstall`, `swconfig`, `swcopy` and `swremove` tasks automatically add to, change and delete IPD and catalog file information as the commands are executed. `swlist` and `swverify` tasks read the IPD information and use it to affect command behavior.

**Note** *You cannot manually edit the IPD or catalog files*, However, the SD-UX `swmodify` command can be used to alter individual pieces of information in the IPD and catalog files (see Chapter 7 for more information).

The IPD also contains a `swlock` file that manages simultaneous read and/or write access to software objects.

2

# Installing or Copying HP-UX Software

As stated in Chapter 1, the `swinstall` command installs or **updates** software from a software source (a depot or physical media) to your local host. The `swcopy` command *copies* software from a source to a depot which can then be used as an installation source. *Software that is copied into a depot cannot be used*; it is placed there only to act as a source for other installations.

Other differences between `swinstall` and `swcopy` include:

- Compatibility checking (that is, making sure the software will run on the installed system) is done for `swinstall`, but not done in `swcopy`.
- Control scripts are not run for `swcopy`.
- Kernel rebuilding or rebooting is not done for `swcopy`. Other pre- and post-install checks, such as disk space analysis and requisite selection, are still performed.
- When a depot is created or modified with `swcopy`, "catalog files" are built that describe the depot (catalog file are similar to the files created in the SD-UX Installed Products Database for products and installations).

A **depot source** may contain software for a variety of machines or architectures (except you cannot intermix HP Series 700 and 800 software on the same depot). `swinstall`'s **compatibility filtering** ensures that the architecture of the products matches the host when that software is installed to the target's default directory (`/`).

Software to be used on the local host is normally installed relative to the root directory (`/`), so it is the SD-UX default "target" location. You can also install to an **alternate root directory** of your choosing (see "Installing to an Alternate Root").

You can also install (with `swcluster`) software on a diskless server, with the software contained in a **shared root** which is then linkinstalled to diskless clients. See the section "Advanced Topics for `swinstall` and `swcopy`" for more information on installing to alternate roots and diskless servers.

**Note** If the SD-UX filesets (that is, the `SW-DIST` product on your Product Media) are corrupted or missing from your system disk, *you cannot install any software packaged in SD format.* If this happens, please refer to Appendix C for instructions on how to load (or re-load) SD-UX from the media using standard HP-UX commands.

SD-UX installed software is automatically configured by `swinstall` to run on the local host only if the software is loaded into the host's default `/` directory or root file system. If the installation is to an alternate root, software configuration is not automatically done, but may be performed later using the `swconfig` command (see Chapter 3).

## Installing or Copying Software Using the Command Line

To start an install or copy session via the command line, assemble any options (if needed), host and source names, and software selections into a command string that might look like this:

`swinstall -p -s` *softsource* `-f` *softlist* `@` *myhost:/mydirectory*

The `@` *myhost:/mydirectory* is optional if you are installing to your local host and default directory. If you have several selections, use the command options to "point" to **input files** that contain lists of software selections (`-f` *softlist*, above), default modifications and other variables (see "Input Files" in Chapter 1 for more information).

The syntax for `swcopy` and `swinstall` is:

`swinstall` [*XToolkit Options*] `[-p] [-i] [-v] [-l] [-r] [-s` *source*]
  `[-x` *option=value*] `[-f` *software file*] `[-X` *option file*]
  `[-C` *session file*] `[-S` *session file*] `[-t` *target file* ]
  [*software selections*] `@` [*host designation*]

`swcopy` [*XToolkit Options*] `[-p] [-i] [-v] [-s` *source*] `[-x` *option=value*]
  `[-f` *software file*] `[-X` *option file*] `[-C` *session file*] `[-S` *session file*]
  `[-t` *target file*] [*software selections*] `@` [*depot designation*]

The `-t`*target file* option lets you list multiple targets to which you could `push` software. A controller host's ability to actively **push** software to multiple *remote targets* applies only to the HP OpenView Software Distributor (HP Product No. B1996AA). You cannot push software to remote hosts with the SD-UX commands. To install or copy software simultaneously to remote hosts (targets), you must purchase the HP OpenView Software Distributor.

Using SD-UX syntax, you can specify:

- which software selections to install or copy (using the `-f` *software file* option or named *software selections*),
- where those selections are located (the `-s` *source name*) and
- what host and/or depot location is to receive them (the @ *host:/depot* designation).

See "Command Options" for a full listing of options used with the `swinstall` and `swcopy` commands.

**Note** In the `swinstall` command, if no *source* is specified, the local host's default depot directory (*/var/spool/sw*) is assumed. If no *host* is named, the system to receive the software is assumed to be the root (*/*) directory on your local host. So, you do not HAVE to use the @ sign and `host[:/directory]` designation if you are operating on the local host and/or default depot directory.

## Using `swinstall` to Update

When you use `swinstall` to update or install newer (later) versions of software that already exist on your system, SD-UX treats the "new" and "old" files, directories and symbolic links in the following way listed below. Please refer to Appendix C for instructions on how to update SD-UX from the media.

**Note** You cannot use a tape device (DAT) on a remote host as a source because you cannot control (change tapes, rewind, etc.) that remote device.

- Regular files and directories on the media:
  - □ If the file or directory to be installed *does not exist* on the target, SD-UX creates it and installs the file (or directory and contents) from the media to the target.

    - If a file with the same name *already exists* on the target, SD-UX overwrites that file with the new one. In this case, previous file contents are lost. Note that the new file may or may not be the same size as the old one.
    - If the object to be loaded is a *file* and a *directory* by the same name already exists on the local host, SD-UX renames the old directory and installs the new file in its place (that is, replacing a *directory* with a *file*).
    - If the object to be loaded is a *directory* and a *file* by the same name already exists on the local host, SD-UX renames the file on the system and creates a directory at that location (that is, replacing a *file* with a *directory*).
- Symbolic links on the media:
    - Symbolic links are shipped unresolved. If the link's target does not exist on the system, the result of installing the link is an unresolved symbolic link.
    - If the symbolic link target is an existing file or directory on the system, the contents of the target file or directory are overwritten. Mode of the link target is set to the mode of the file on the source media. Owner and group of the link target are unchanged.
    - If a directory link's target is a file on the system, SD-UX renames the file into a directory at the target path.
    - If a file link's target is a directory on the system, SD-UX removes the target directory and its contents and installs the new file to the target path.
    - Linking to another symbolic link - if the link is unresolved or resolves to a file or directory it is handled as above.

### Symbolic Link Loops

A *pre-existing* symbolic link loop will cause an error message in the `swinstall` process regardless of the path it takes over hard mounted volumes. However, if the loop is the result of installing another symbolic link, there will be no error until a process attempts to write to it. File load order is important in determining if a symbolic link loop will cause a problem with SD-UX operations.

For example, suppose you have a file called *Alpha* on your system and *Beta* is a symbolic link to *Alpha*. During an update, you attempt to install a new file called *Beta* that has a link to it called *Alpha*. *Alpha* is loaded from the media first. Since *Alpha* is a symbolic link to *Beta*, the file at *Alpha* is overwritten and there is now a loop. When the new *Beta* is loaded, it fails, having encountered the loop.

Make sure the load order on your update does not cause these link loops and also make sure that links are not "pointing" to each other after the update.

## Sample Install/Copy Operations

### Installing

1. To install a pre-determined list of software products in the file *mysoft* that are physically on a CD-ROM (mounted locally at `/mnt/cd`) to the default directory (`/`) on the local host:

   ```
   swinstall -f mysoft -s /mnt/cd
   ```

   Note: The @ sign was not used because it is *assumed* that the installation will be to the default directory on the local host.

2. To pull all the software selections that are in the default depot (`/var/spool/sw`) located on a host named *server* to the default directory on your local host named *myhost* and preview the process (`-p`) before actually installing:

   ```
   swinstall -p -s server \* @ myhost
   ```

   We did not specify the depot location (`:/`*depot*) because it is assumed that the software is located in the default `/var/spool/sw` on *server* and will be installed at `/` on *myhost*. The `-p` analysis option is explained below under "Command Options." The `\*` is an optional shorthand wildcard meaning "all products and filesets or all available software."

3. To select all the products named "C" and "Pascal" from the default depot on the host named *sw_server* and start an interactive GUI session (`-i`):

   ```
   swinstall -i -s sw_server C Pascal
   ```

4. To update HP Omniback software (already installed in the default directory on the local host) with a newer version from a CD-ROM mounted at `/mnt/cd`:

   ```
   swinstall -s /mnt/cd Omniback
   ```

   "Updating" in SD-UX means that you are completely overwriting the old version filesets with the new (see "Using `swinstall` to Update").

### Copying

1. To copy all products from the DAT tape at `/dev/rmt/0m` to the default depot (*/var/spool/sw*) on the local host:

   ```
   swcopy -s /dev/rmt/0m \*
   ```

2. To copy a list of software selections (on a local CD-ROM) named in the file *mysoft* to a depot at the path */var/spool/sw* on the host named *hostA* and preview the process before actually copying the software:

   ```
   swcopy -p -f mysoft -s /mnt/cd @ hostA:/var/spool/sw
   ```

## Command Options

Many of the options listed here for `swinstall` and `swcopy` are the same for other SD-UX commands.

| **Option** | **Action** |
|---|---|
| `-p` | Preview an install or copy task from the command line by running it through the Analysis Phase and then exiting. This option can be used with any of the other options to understand the impact of a command before the system actually does it. |
| `-i` | Run the command in interactive mode with "pre-specified" software selections. Displays the first Graphical or Terminal Interface Window. |
| `-v` | Turn on verbose output to `stdout` and display all activity to the screen. Lets you see the results of the command line activity while it is being performed. |
| `-l` | Runs the command in **linkinstall** mode that makes software, already installed under a diskless server's shared root, available to a diskless client. There is no `-l` (linkinstall) option for `swcopy`. |
| `-r` | (Optional) Specifies alternate root directories. See "Advanced Topics for `swinstall` and `swcopy`" for more information about installing to alternate roots. There is no `-r` option for `swcopy`. |
| `-s` *source* | Specify which software source is to be used for the installation or copy. For an installation, the default source type is a directory or depot (usually */var/spool/sw*) on |

| | |
|---|---|
| | the local host, not a tape device. The default source for linkinstall is the shared root */export/shared_roots/OS_700*. |
| `-x` *option=value* | Lets you change a default value on the command line that overrides its value or a value in an options file (`-X` *option file*). See the section "Advanced Topics for `swinstall` and `swcopy`" for more information on defaults. |
| `-f` *software file* | Read a list of software selections from a separate file instead of from the command line. In this software file, blank lines and lines beginning with # (comments) are ignored. Each software selection must be specified on a separate line. For an example of a software selection file, see section "Command Line Software Selections". |
| `-t` *target file* | This option is more applicable to the HP OpenView Software Distributor product that allows installing to multiple remote hosts (targets). `-t` reads a list of these targets from a separate file instead of from the command line. You could, however, use it to specify multiple *shared roots* on the local host. |
| `-X` *option file* | Specifies a new option file. The default values for system options are provided in the file */var/adm/sw/defaults*. You can also provide a personal option file, *$HOME/.sw/defaults*. This option file overrides those values in the system defaults file. For a complete listing of system options, see the file */usr/lib/sw/sys.defaults*. This file lists the possible values and behaviors for each option for each command. See also the section "Advanced Topics for `swinstall` and `swcopy`" for more information on options and defaults. |
| `-C` *session file* | Run the command and save the current option and operand values to a file for re-use in another session. |
| `-S` *session file* | Run the command based on values saved from a previous installation session. |

## Command Operands

Software selections and source/target designations must be named using a particular syntax and format so SD-UX can identify and locate them.

### Command Line Software Selections

SD-UX lets you describe software objects in very general or very specific terms. You can describe an object as a bundle, product, subproduct, or fileset:

| | |
|---|---|
| **Bundle** | A collection of filesets encapsulated for a specific purpose. |
| **Product** | Collections of subproducts and filesets. |
| **Subproduct** | An optional grouping of filesets (used to partition a product that contains many filesets or to offer the user different views of the filesets). |
| **Fileset** | A collection of files. |

If you specify a bundle, product, or subproduct, SD-UX automatically includes all filesets under that bundle, product, or subproduct. Every software specification must include a bundle or product or both.

Bundles can only be packaged by HP at this time; so when you specify a bundle for operations other than packaging, you will see only those products and filesets that have been loaded from (or exist on) the HP-UX OS media. The default software "view" is "all_bundles" (that is the *software_view=* default is set to *all_bundles*) which shows bundles plus top level products that are not part of any bundle.

Software selections are specified using the following syntax:

```
bundle[.]
```
*or*
```
product[.subproduct][.fileset][,version]
```

Specifying `bundle.` (bundle dot) signifies an empty bundle that is processed without reference to its contents. This allows a bundle to be operated upon without affecting its contents.

The *version* component is always preceded by a comma instead of a period and can be further specified as:

```
,[r=revision],[a=architecture],[v=vendor],[c=category],[instance_id]
```

Each version specification (`r, a, v, c,` etc. above) is repeatable within the component and must also be separated by a comma.

The **instance_id** specification is a number you can assign (in the IPD for the product) to further distinguish between similar versions; for example, *Mysoft,3* would be the third version of the product *Mysoft*. This form should only be used when you are sure of the product contents.

To give you an idea of how precise your software selection can be, here is a sample software selection:

```
SoftwareA.subproductAa.filesetAA,r<=C,a=HP-UX
```

This would describe the HP-UX version (revision C or earlier) of *filesetAA* of *subproductAa* of a product called *SoftwareA*. This is a specific designation of an individual fileset within a subproduct in a product.

On the other hand, you might just specify:

`BundleX`
*or*
`ProductY`

and get all the products and filesets previously packaged under them without having to specify them individually.

## Command Line Source and Target Designation

Hosts and sources can be specified by hostname, hostname alias, or full network address, optionally followed by an alternate root or depot directory path. For example, all of the following inputs are valid as host identifiers:

```
hphost1
hphost1.fc.hp.com
15.10.40.33
hphost1:/alternate_directory
/alternate_directory
```

You may omit the hostname or address if your target is the local host.

During normal operation, if you do not specify a source location or directory, the default depot on your local host is always assumed. For linkinstall operations, the local host's shared root (*/export/shared_roots/OS_700*) is assumed. If no host is specified, the root directory on the local host or the default depot as listed in the defaults file is assumed.

A typical host or source input file might look like this:

```
hostA:/directoryF
#hostA is Roger's machine, directoryF is an alternate root where
#his software is available.

15.15.232.25
#This is Janet's machine, identified by her network address.

hostC.upi.com
#This is Mark's Internet address.
```

Comment lines are designated with a `#`.

## Changing Options and Defaults

In addition to the command-line options listed above, the */usr/lib/sw/sys.defaults* template file lists and explains nearly sixty different SD-UX command options, their possible values, and the resulting system behavior. These options are listed as comments that you can copy into the system defaults file (*/var/adm/sw/defaults*) or your personal defaults file (*$HOME/.sw.defaults*).

Each value (known as a **default option value**)in this file is specified using the `command.option=value` syntax. For example:

```
swinstall.allow_downdate=false
```

If you placed this line in your */var/adm/sw/defaults* (for system-wide control) or *$HOME/.sw.defaults* file (for individual user control) and changed the value from false to true, the system would allow installation of older versions over newer versions *for all future sessions*. (This is called **downdating** as opposed to updating).

These rules govern the way the defaults work:

1. Defaults in */usr/lib/sw/sys.defaults* are hard-coded and are used only when there are no other default files or session files that contain different values.
2. Defaults in */var/adm/sw/defaults* file affect all users in a system.
3. Defaults in your personal *$HOME/.sw/defaults* file affect only you and not the entire system.
4. Defaults in a session file affect activities only for that session and revert when that session is completed.
5. Defaults changed on the command line affect only that activity.

For system-wide policy setting, use the */var/adm/sw/defaults* files. Keep in mind, however, that individual users may override these defaults with their own *$HOME/.sw/defaults* file. Defaults can also be overridden by session files or command line changes.

To change a default value or behavior for an SD-UX command, simply copy the specific default line (the un-commented line) from the file */usr/lib/sw/sys.defaults*, add it to the file */var/adm/sw/defaults* (for system-wide changes) or *$HOME/.sw/defaults* (for individual user changes) and then change its value.

See "Advanced Topics for `swinstall` and `swcopy`" for more information on how to modify these defaults and options. Appendix A and the file */usr/lib/sw/sys.defaults* have a complete list of all system default options, their values and a brief description of their effects.

## Using Software Codewords and Customer IDs

To protect software from from unauthorized installation, HP (and other vendors) use special codewords and customer identification numbers to "lock" the software to a particular owner. These codewords and customer IDs are provided to you when you purchase the software (or receive it as update). HP lists them on the Software Certificate which is packaged with the software.

A codeword for a particular customer ID and CD-ROM only needs to be entered once per target system. The codeword and customer ID are stored for future reference in the */var/adm/sw/.codewords* file.

SD-UX will prompt you for these codewords or numbers prior to the installation of protected software. You can enter or change the numbers via the Graphical User Interface (using the `Add New Codeword` choice from the `Actions` menu in the GUI) or by using the appropriate default (*-x codeword=xxxx* and *-x customer_id=xxx*) on the command line. See Appendix A for more information on codeword and customer_id defaults.

**Note**

For HP-UX B.10.10 and later systems, SD searches the *.codewords* file on the server that is providing protected software to other hosts. It looks for valid customer_id/codeword pairs. In doing so, SD eliminates the need to enter codewords and customer_ids on every host that is "pulling" the software.

To properly store the customer_id/codeword for a CD-ROM, run `swinstall` or `swcopy` on the host serving the CD-ROM. After the codeword has been stored, clients installing or copying software using that host and CD-ROM as a source will no longer require a codeword or customer_id.

This is a timesaver if you are updating multiple systems.

For example, if you want to store the codeword *123456789101bcdf* (from the */CD-ROM* mount point) and your customer_id was *xyzCorp*, you would enter on the command line:

```
swinstall -x customer_id=xyzCorp -x codeword=123456789101bcdf -s /CD_ROM
```

Error messages can be ignored since you are only storing codewords and customer_ids.

---

## Recovering Updated Files

If you should start an install operation and, in the middle, the process fails, in certain instances SD-UX may allow you to automatically recover or "rollback" to the original product files.

Placing the *swinstall.autorecover_product=true* default line in */var/adm/sw/defaults* or *$HOME/.sw/defaults* lets you recover product files that are normally removed as they are updated. If you encounter an error while loading new filesets, the product being loaded will be marked CORRUPT and you must re-try the installation.

---

**Note** Due to the complexities of the preinstall and postinstall scripts associated with all products delivered as part of HP-UX 10.X, "autorecovery" of these products is not supported. All HP-UX 10.X products have preinstall and postinstall scripts without accompanying "undo" scripts. Consequently, do not change the *autorecover_products* option to "true" prior to installing any HP-UX software.

If the software uses no preinstall or postinstall scripts or has the appropriate "undo" scripts, "autorecover" will work.

---

By setting the *autorecover_product=* option to "true," all files that are removed will be saved as backup copies until all filesets in the product have completed loading. This allows automatic recovery of all filesets if the load fails.

If *autorecover_product* is set to "false" (the default value), all files that have already been removed cannot be recovered.

## Installing/Copying Software with the Graphical or Terminal User Interface

This section provides a quick overview of the Graphical User Interface (GUI) and Terminal User Interface (TUI) features, pointing out the major capabilities. It is not a step-by-step procedure guide or road-map. As you use the GUI or TUI, you will become more familiar with the processes and menu items. If you have questions about specific menu choices, use the On-Line Help feature (place the cursor/focus on the item and press `f1` on your keyboard) for more information.

You can also press the `Help` button in the Menubar or the button at the lower right of the screen for information on the whole screen.

If you put `/usr/sbin` in your PATH, you can dispense with the `/usr/sbin` path prefix. The TUI is the default mode of interacting with `swinstall, swcopy` and `swremove` *if you have NOT set your DISPLAY variable.* If you *have* set your DISPLAY variable, invoking these commands will automatically start the GUI.

To start the Graphical or Terminal User Interface for an install or copy session, type:

```
/usr/sbin/swinstall
   or
/usr/sbin/swcopy
```

Open a `swinstall` session now on your system so you can follow along with the manual and try out the menus and dialogs.

There are three phases in the install or copy process:

**Selection** Specifying the software you want to install or copy (software selections) and where that software is located (source) and where it is to be installed or copied (target). You can also change default options in this phase and specify target paths for `swcopy` and installations to alternate roots. The SD-UX GUI and TUI offer two interactive screens for this phase: the Source Selection Dialog and the Software Selection Screen.

**Analysis** An analysis of the task is performed BEFORE actually installing or copying the software to make sure the process will be successful. The process is monitored and you can see the results as they occur. SD-UX provides an Analysis Dialog that overlays the Software Selection screen for this phase.

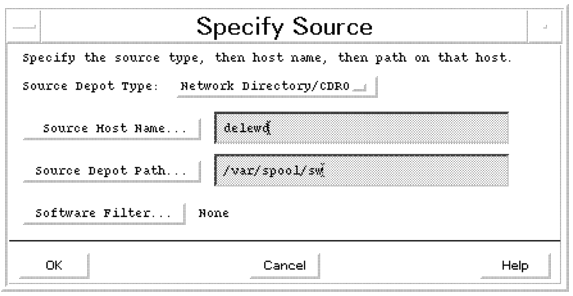
**Load/Execution**

The actual software installation or copy and further monitoring of the process. SD-UX provides a separate Install/Copy Dialog for this phase. For `swinstall`, any install and configure scripts are executed and kernel building or rebooting is done at this time, as needed. See Appendix B for more information on control scripts.

`swinstall` and `swcopy` also automatically save the current command options, source information, software selections and hosts into a **session file** before the operation actually starts. This session file can then be recalled and re-executed. This session file is saved as *$HOME/.sw/sessions/swinstall (swcopy).last* (see "Managing Sessions - The `File` Menu" for more information).

## Selecting the Source

When you type `swinstall` or `swcopy`, the Software Selection Window appears on your screen with the Specify Source Dialog superimposed over it. The dialog automatically lists the local host and default depot path, but you can also specify some other the host and depot where the software is located. You can also choose what level of software to "view": All Bundles, Products, or Bundles of One Category.



```
Specify the source type, then host name, then path on that host.
Source Depot Type:    Network Directory/CDRO
Source Host Name...   delewd
Source Depot Path...  /var/spool/sw
Software Filter...    None
```

**Figure 2-1. Specify Source Dialog**

### Adding Sources

If you click on the `Source Host Name` button in the Specify Source Dialog, the system will display a box with all the *hosts* that are in your *defaults.hosts* file (`$HOME/.sw/defaults.hosts` or `/var/adm/sw/defaults.hosts`). You can highlight (choose) one of the hosts and press `OK` and it will appear in the appropriate box in the Specify Source Dialog. You can also manually enter a source host name in the text box next to the `Source Host Name` button if the source you want does not appear in the dialog.

Here is a sample `$HOME/.sw/defaults.hosts` file:

```
swinstall.hosts = {hostA hostB hostC}
swinstall.hosts_with_depots = {host-depotA host-depotB}

swcopy.hosts_with_depots = {SourceA SourceB}

swremove.hosts = {SourceX SourceY SourceZ}
swremove.hosts_with_depots = {Source1 Source2 Source3}
```

Multiple host names must be separated by a space. You must edit this file manually to add or delete hosts and depots. If your list of hosts will fit on one line, you do not need to delimit the list with brackets ({}).

If there are no hosts specified in the *defaults.hosts* file, only the local host and default depot path appear in the object lists.

Clicking on the `Source Depot Path` button will bring up a list of registered depots on the source host. You can highlight one of the depots and press `OK` and it will appear in the Specify Source Dialog. You can also manually enter a depot path in the text box next to the button.

### Adding Depots

Selecting depots for copying involves a source host and target depot path pair. For example, on the command line, depots are specified with the syntax `host:/depot_path`. The same is true in the GUI/TUI.

When a target depot is specified, if it does not exist, the system will create a new depot.

**Note** Series 700 and Series 800 software cannot coexist on the same depot. If you have software for both these platforms you should create separate depots for each.

### Managing Sessions - The `File` Menu

The Menubar for the Software Selection Window has five activities: `File View Options Actions Help`. Each of these choices provide additional "pulldown" menus for more activities. Placing your mouse cursor on the appropriate Menu choice and "clicking" the left mouse button causes additional menus to appear.

As we have said, each invocation of a command is considered a "session". When you invoke a command, it automatically saves the current command options, source information, software selections and host designation before the task actually starts. The `File` Menu, which is common to all GUI windows, manages these session files.

A session file, which is used in the `-C` or `-S` session file option described in the command line syntax, can be recalled and re-executed if you often use the same functionality. All selections are always saved in *$HOME/.sw/sessions/<swcommand>.last* file just before starting the actual installation.

At the end of each session the system will overwrite the previous session file. If you want to save a session file for use later, you must rename that session file.

Here is an example of a session file:

```
# swinstall session file
#
# Filename      /users/fred/.sw/sessions/swinstall.last
# Date saved    05/26/93 15:59:41 MDT

swinstall.allow_downdate = true
swinstall.allow_incompatible = false
swinstall.allow_multiple_versions = false
swinstall.autoreboot = false
swinstall.autorecover_product = false
swinstall.compress_files = false
swinstall.create_target_path = true
.
.
```

The session file uses the same syntax as the defaults files (*command.option=value*). Session file values are, in turn, overridden by the corresponding command line options, if they are specified. See "Advanced Topics for `swinstall` and `swcopy`" for more information on defaults.

For a description of each menu choice in the Menubar, click on a specific menu choice or use the arrow keys to highlight a particular menu choice and press the [f1] key on your keyboard. This brings up a Help Screen that provides additional information on that menu choice.

## Changing Preferences - The `View` Menu

The Graphical and Terminal User Interfaces can be modified to fit your individual display requirements or view preferences. The `View` menu manages these view preferences by providing a Columns Editor to "customize" the window by changing the names and arrangement of the items in the Object List. The Columns Editor is accessed through the `View`→`Columns...` menu choice. Changes you make can also be saved (`View`→`Save View as Default`) and are treated as the default the next time you invoke the command.

You can change your Software View (in the Software Selection Window) with the `Change Software View` menu choice or the `Change Software Filter` button. You can specify Bundle, Product or Bundle Category views (that is, which "level" of the software object hierarchy you want to display at the topmost level). You can also sort your list of selections in a variety of different ways. Sorting is specified in a separate `Sort` Dialog.

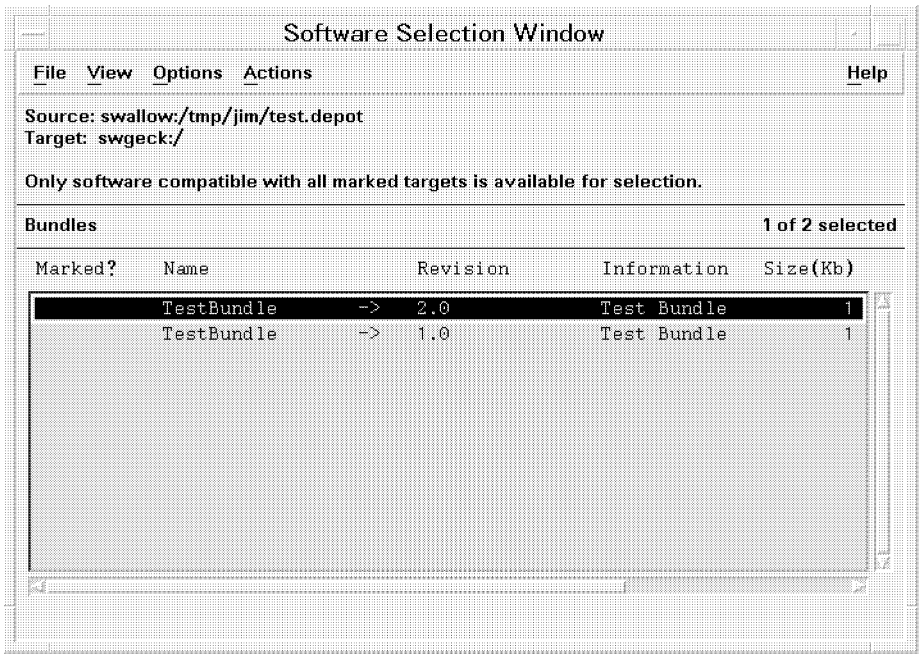
### Changing Options - The `Options` Menu

The `Options` menu lets you change the default options file that controls command behaviors and policies. This is the same as editing the defaults options file `/var/adm/sw/defaults` or *$HOME/.sw/defaults* . For a complete description of the Options Editor and how to change options with this dialog, see the section "Advanced Topics for `swinstall` and `swcopy`".

### Choosing Software - The `Actions` Menu

When you have specified the host, source and depot (or shared_root), you are ready to select software to install or copy. Software is selected from the Object List by highlighting items and then invoking the `Mark For Install` (or Copy ) menu choice. In the TUI, you can also use the "m" and "u" keys to mark and unmark highlighted objects.

The Marked? flag in the Object List is automatically updated to "Yes" to reflect the selection. An additional flag - "Partial" - is provided to show that only some component of the software selection has been selected for the install.



Source: swallow:/tmp/jim/test.depot
Target: swgeck:/

Only software compatible with all marked targets is available for selection.

Bundles 1 of 2 selected

```
Marked?   Name              Revision      Information   Size(Kb)
          TestBundle    ->  2.0           Test Bundle          1
          TestBundle    ->  1.0           Test Bundle          1
```

**Figure 2-2. Software Selection Window**

**Note** Only one version of a product can be selected during a single installation session.

Once software is selected, the system then provides some additional capabilities (via the `Actions` menu choice):

- `Match-What-Target-Has` examines your current Installed Product Database to match your existing filesets with new filesets (those with the same names) that you are going to install. This feature is most helpful when you are updating a system to newer versions of the same software. This feature is controlled by the *match_target*= option (see Appendix A).
- `Change Source` prompts you for a new source selection.

  If you elect to pick a new source, a Change Source Dialog Box is automatically displayed so you can specify another source. You can type it in or select it from the host and path lists.

- `Install (analysis)` (or `Copy (analysis)`) launches the next phase (analysis).

Also in the `Actions` menu:

- `Open Item` or `Close Level` allows you to see the contents of a selected object or close it. In the TUI, pressing "Return" opens a highlighted object.
- `Show Description of Software` displays more information on the selected software.
- `Mark` or `Unmark` marks or unmarks software for an operation.
- `Change Product Location` lets you revise the software source.
- `Add New Codeword` lets you add a new codeword. (This option is available if SD-UX detects that the source contains protected software.

## Opening a Software Selection

The Software Selection Window Object List is a "hierarchical" object list. That is, each object in the list may be "opened" to show its' contents. For example, to see the subproducts in a particular product, you can "open" that product by "double clicking" on the object with your mouse. In the TUI, move the cursor to the item you want to open and press `Return`. The Object List then shows a listing of the subproducts for that product. If you want to open the subproduct, you would double click on it and its filesets would be displayed. Objects that have an ->▸ after the name can be opened to reveal other items.

To close an object and return to the previous list, double click on the first item in the list (`. .(go up)`). In the TUI, you must use `Close Level` in the Actions menu or press "Return" while highlighting the (`. .(go up)`) item.

When the product is opened, all of its subproducts (and filesets that are not part of a subproduct) are shown in the list. At the product level, only products are listed together. If the *software_view* is "Bundle" and the bundle is opened, all HP-UX OS products that are wholly or partially contained in the bundle will be shown. When one of the products is opened, only subproducts and filesets in the open product and open bundle are shown.

## Getting More Information

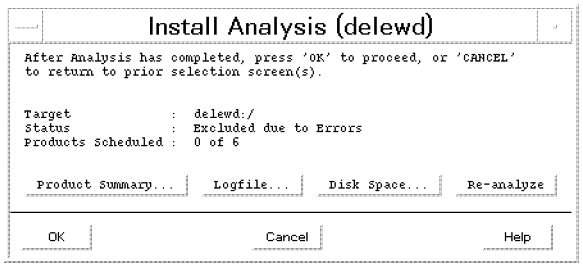
The first line of the Message Area is used to show which source has been chosen. The second line of the Message Area displays where the software will be installed or copied using a `host:/depot` syntax. These lines can be modified with the `Change Source` menu choice under the Actions menu. If an error occurs when the source is read, the Change Source Dialog is automatically displayed for input.

The Object List shows the software's "attributes" (Name, Revision, Information , Size (Kb), Architecture and Category). The `Show Software Description` menu choice also provides more information.

Choosing `Install analysis...` (or `Copy - analysis`) in the menu moves you to the next phase of the process by opening the Install (or Copy) Analysis Dialog. If software selections or host selections have not been made, an error box gives instructions on what items still need to be selected before the Analysis Dialog appears. The Software Selection Window remains open so you can examine your selections at any time. Canceling the Analysis will also return you this window to change selections.

## Analyzing Your Installation Or Copy

When the Analysis Dialog appears, analysis checks are automatically started on the selected target and source.



```
After Analysis has completed, press 'OK' to proceed, or 'CANCEL'
to return to prior selection screen(s).

Target             :  delewd:/
Status             :  Excluded due to Errors
Products Scheduled :  0 of 6
```

**Figure 2-3. Analysis Dialog**

The buttons in the Analysis Dialog let you investigate the logfile for details on the process, display more information about a product, check disk space analysis or restart the analysis.

### Analysis Progress and Results

The Status column in the Analysis Dialog shows the progress and the results of the Analysis Phase. The possible values for the Status are: "Analyzing Targets," "Analyzing Software," "Finishing Analysis" and "Ready".

When the Analysis is done, the status for any host will either be "Ready" or "Excluded from task". If *any* of the selected software can be installed onto the host, the status will be "Ready." If *none* of the selected software can be installed onto the host, status will be "Excluded from task".

Here is a summary of the status results:

| | |
|---|---|
| Ready | There were no errors or warnings during analysis. The installation or copy may proceed without problems. |
| Ready with Warnings | Warnings were generated during the analysis. Errors and warnings are logged in the logfile. To see the logfile, press the `Logfile` button. |
| Ready with Errors | At least one product selected will be installed or copied. However, one or more products selected are excluded from the task because of analysis errors. Errors and warnings are logged in the logfile. To see the logfile, press the `Logfile` button. |
| Communication failure | Contact or communication with the intended target or source has been lost. |
| Excluded due to errors | Some kind of global error has occurred. For example, the system might not be able to mount the file system. See the Logfile for details. |
| Disk Space Failure | The installation (or copy ) will exceed the space available on the intended disk storage. For more details, press the `Disk Space ..` button. |

The Products Ready column shows the number of products ready for installation or copy out of all products selected. These include:

- products selected only because of dependencies
- partially selected products
- other products and bundles that were selected.

See the section "Advanced Topics for `swinstall` and `swcopy`" for more information regarding dependencies.

Only "ready" products are shown. Bundles may also be ready. If they are, their products are reflected in the product contents and list.

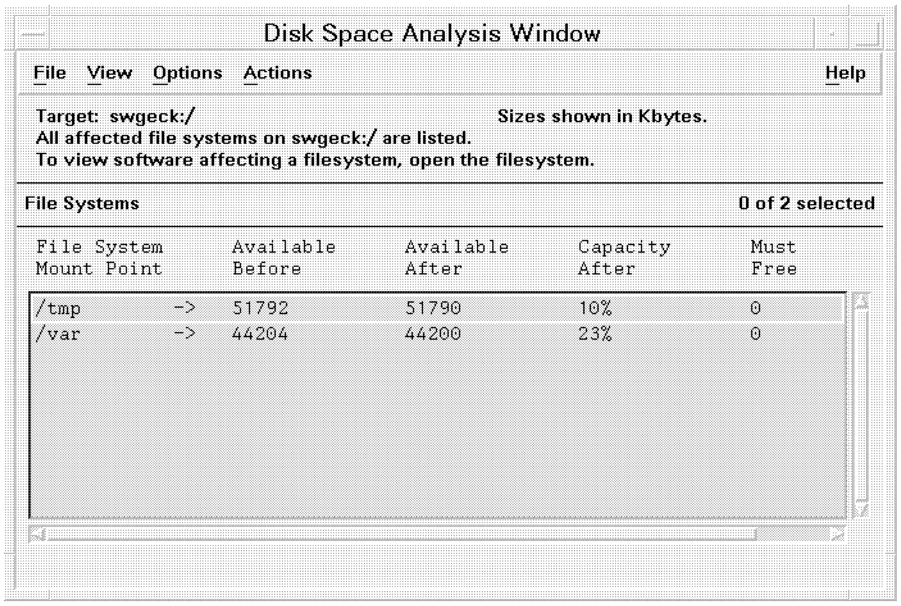
A product may be automatically excluded from the task if an error occurs with the product. The host is automatically excluded from the task only if ALL products and bundles have been excluded.

## Getting More Details

The `Product Summary` button in the Analysis Dialog gives additional information regarding the product (or bundle) and provides a `Product Description` button that displays information about dependencies, copyright, vendor, etc.

The `Logfile` button presents a scrollable view of the information that is written in the logfile.

The `Disk Space` button shows the file system mount point, how much disk space was available before the installation, how much will be available after the operation, and what percent of the disk's capacity will be used. It also shows how much space must be freed to complete the operation.



Target: swgeck:/ Sizes shown in Kbytes.
All affected file systems on swgeck:/ are listed.
To view software affecting a filesystem, open the filesystem.

File Systems 0 of 2 selected

```
File System       Available      Available      Capacity      Must
Mount Point       Before         After          After         Free
/tmp        ->    51792          51790          10%           0
/var        ->    44204          44200          23%           0
```

**Figure 2-4. Disk Space Analysis Dialog**

## Projected Actions

When you press the `Product Summary` button in the Analysis Dialog another dialog appears that gives you additional information about the product and the operation. The Projected Actions column describes what type of installation or copy is being done. All actions are listed if more than one applies. For example, a product may be a New Install (Copy) and an Update (some filesets are new, some are an update). The possible types are:

| | |
|---|---|
| New Install (Copy) | The product does not currently exist on the host. This product will be a new installation or copy. |
| Reinstall (Recopy) | Some parts of the product already exist on the host (at the configured location). Note that this effect is only possible if *reinstall=true*. Otherwise, this same product would be skipped. See Appendix A for more information on the *reinstall* option. |

| | |
|---|---|
| Update | An older version of the product already exists on the host (at the configured location) and the selected component will be an update to the existing product. |
| Downdate (allow previous versions) | A newer version of the product already exists on the host (at the configured location) and the selected product will be a older version of the existing product. Note that this effect is only possible if *allow_downdate*= is set to "true." Otherwise, this same product would be excluded because of a downdate error. See the section Appendix A for more information on the *allow_downdate* option. |
| Multiple Version | No component of the product currently exists on the host at the configured location, but some version of the product does exist at some other location. If installed, this product creates a multiple version of the product on the host. (Note that this effect is only possible if the option to *allow_multiple_versions* is "true." Otherwise, this same product would be excluded because of a multiple version error.) See section "Advanced Topics for `swinstall` and `swcopy`" for more information on the *allow_multiple_versions* option. |
| Skipped | The product will not be installed or copied because of a condition that is NOT considered an ERROR. For example, the same product may already be installed on the target (and *reinstall=false*) |
| Excluded | The product will not be installed because of some analysis phase errors. See the logfile for details about the error. Current reasons a product would be excluded from the installation are:<br>■ the checkinstall script failed for the product or a fileset in the product<br>■ installing the product at the configured location would create a multiple version and *allow_multiple_versions* is set to "false."<br>■ the product depends on another product that has also been excluded because of errors, or was not selected and is not installed. See the section "Advanced Topics for `swinstall` and `swcopy`" for more information about dependencies. |

If a global error occurs, e.g., a disk space error occurs or file systems can't be mounted, all products would be excluded. After the installation the Result will show:

| | |
|---|---|
| Installed | The product was successfully installed, but not automatically configured. |
| Configured | The product was successfully installed AND configured. |
| Corrupt | Because of errors during the execution phase, the product was not completely installed. The product is now corrupt and cannot be used. |

After a copy has completed, the Copy Summary also displays the "state" of the product - AVAILABLE or CORRUPT. The AVAILABLE state is the most common.

The Summary column shows any warnings or errors that occurred.

## Seeing the Logfile

The Logfile contains detailed information about the installation or copy process. If the current host is still in Analysis Phase, the logfile is updated periodically with the in-coming results. Pressing the `Logfile` button in the Analysis Dialog displays the Logfile.

Moving the scrollbar at the right side of the **Logfile Dialog** controls scrolling. By default, when the Logfile Dialog is first opened, `Automatic scrolling` is toggled on and the text of the logfile continually scrolls up as new information is added. The old information scrolls off the top of the list while the new information is added to the bottom of the list. You may turn automatic scrolling off by clicking on the `Automatic scrolling` button or by moving the scrollbar),

When automatic scrolling is turned off, the new information is still added to the bottom of the list but the list itself does not scroll. In this way, you can move to a particular place in the logfile and not be bothered by the automatic scrolling.

## Analyzing Disk Space

The `Disk Space` button in the Analysis Dialog brings up a Disk Space Analysis Window that displays the status of the file systems on your disk. It shows the file system, the amount of space available on the disk BEFORE you attempted to load the software, how much room will be available AFTER the installation is completed, the capacity remaining, and how much free disk space is required (if any).

Opening the software (via the `Open Item` menu choice in the `Actions` menu) and double clicking on one of the filesets in the software will tell you the projected size of fileset on this file system and what the change will be in + or - Kbytes.

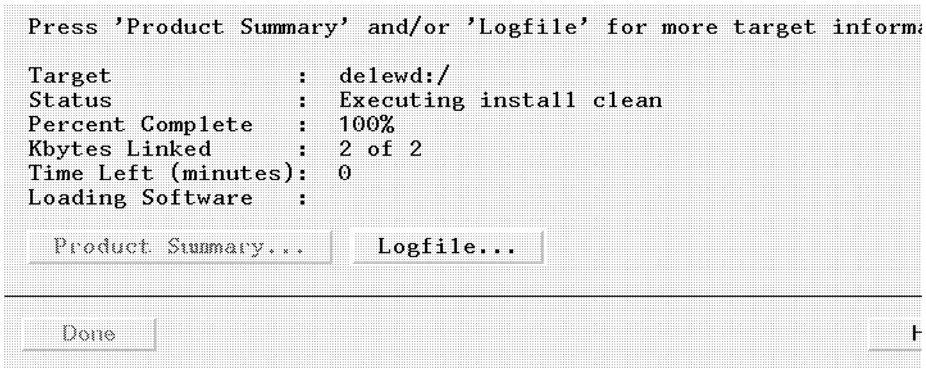
## Installing/Copying the Product - the Install (Copy) Dialog

If the host has a status of "Ready," you can press `OK` to start the installation or copy. The Analysis Dialog is then replaced by the Install (Copy ) Dialog and a confirmation dialog appears explaining that once the load is started, you cannot return to the Selection or the Analysis phase. If you say "No" in the Confirmation Dialog, you will return to the analysis. The installation or copy has not started and no Install or Copy Dialog is presented.

Before the Copy phase is started and if you are copying from a tape, a check is made to detect if the source tape will need to be changed during the phase. If the source tape must be changed, a Final Copy Dialog notifies you.

The Install or Copy phase may be suspended if an error occurs in a script which causes the load to be suspended. For this situation, the `Resume Copy/Install` and `Abort Copy/Install` menu items are available. There are also `Logfile` and `Product Summary` buttons so you can see the results. The resume and abort buttons are not shown when the functions are not available.

Only the host on which software is being installed or copied is displayed in the Install (Copy) Window.



```
Press 'Product Summary' and/or 'Logfile' for more target informa

Target              :  delewd:/
Status              :  Executing install clean
Percent Complete    :  100%
Kbytes Linked       :  2 of 2
Time Left (minutes):  0
Loading Software    :
```

**Figure 2-5. Install Dialog**

The `Resume ..` button allows you to continue the install process if it is suspended. The install phase may suspend if:

- a tape change is needed (for multi-tape media),
- file loading fails,
- customization for kernel-related filesets fails or
- building a new kernel fails.

If the install phase suspends on a host, fix the problem(s) and continue the install phase.

## Monitoring the Install or Copy Process

The Status row displays the progress and the results of the phase. The possible Status attributes and their meanings are:

**Table 2-1. Install Status**

| Status | Meaning |
|---|---|
| Completed | No errors or warnings |
| Completed with Warnings | Warnings |
| Completed with Errors | Errors |
| User Aborted | User Initiated |
| See Logfile | Details are available on Logfile |

The Object List displays other install status attributes:

- The name of the target.
- The name of the software currently being loaded.
- The estimated time (in minutes) remaining to complete the Install Phase.
- The total number of Kbytes already installed.

If the Install (Copy) Phase is suspended on a host, a warning dialog is displayed to notify you that the install has been suspended.

When the Install (Copy) Phase completes on the host, a dialog is displayed to notify you that the task is completed. At this point, you may:

- review all selection windows,
- display details on any target host,
- save the session before exiting,
- start a new session,
- recall a session or
- exit.

## Advanced Topics for `swinstall` and `swcopy`

The topics discussed in this section are for those who have a firm understanding of the SD-UX commands and want more control over their installation or copy process.

### Installing to an Alternate Root

Software is usually installed relative to the "primary" root directory (*/*) but it can also be installed relative to an alternate root directory.

Installing to an alternate root is also used to set up a shared root source to be used later for installing clients with HP's System Administration Manager (SAM).

The automatic configuration and compatibility filtering that is part of the `swinstall` command is not performed when installing to an alternate root.

### Installing/Updating Software on a Cluster Server

To install or update software on a cluster server, a shared root must first exist on the server before clients can be added to create a cluster. This shared root is created by the `Install OS for Clients` menu choice in HP's System Administration Manager (SAM) or by using `swinstall` to install to an alternate (i.e. shared) root.

For more details, see the "Setting Up and Administering an HP-UX Diskless Cluster" chapter in the *HP-UX System Administration Tasks* (B2355-90079) manual.

If you are adding clients to a cluster (that is, making already installed software available to additional clients), you can use the "Install" step performed by SAM or the `swcluster -n` option. A linkinstall sets up read-only NFS mounts to directories in the shared root that contains sharable files such as executables (for example, */usr*) and copies system configuration files and other modifiable files from the shared root to the client's private root (for example, files in */etc* and */dev*).

If no clients are using the shared root, then installation to it is done by using `swinstall` (installing to an alternate root). If the shared root is being used by clients, then installation should be done using the `swcluster` command because it does the alternate root install and takes care of doing the linkinstall for each client. `swcluster` also has a "remove" mode (`-r`) which can be used to remove non-kernel software from a shared root and perform the associated unconfigure steps for each client.

### Using the `swcluster` Command

The `swcluster` command installs (or removes) software from a shared root used by NFS diskless clients. To install software, `swcluster` does an alternate root install to the shared root, then it performs a linkinstall for each client using the shared root. The linkinstall sets up appropriate NFS mounts, copies system configuration files to each client's private root and configures the software on each client. To remove software the reverse is done. `swcluster` can only be run on an NFS diskless server, not on a client.

`swcluster` can also list the various parameters of a server via its "list mode" (`-l`).

If the `-n` option is used, `swcluster` will skip the actual installation of the software to the diskless server. This is useful, for example, if new software is already installed on the server and simply needs to be propagated to the clients.

For example, if you wanted to update the operating system on Series 700 cluster using verbose output (`-vvv` option) and interactive mode (`-i` option) for source and software selection, you would:

1. Invoke the `swcluster` command:

   ```
   swcluster -vvv -i
   ```

   This launches the `swinstall` GUI or TUI.
2. Select the default shared root (*/export/shared_roots/OS_700*) in the interactive UI.
3. Select the software source host and depot.
4. Select `Match what Target Has` from the Actions menu.
5. Select `Install/Analysis` from the Actions menu.
6. After the analysis phase finishes, select `OK` to continue with the cluster install process.

**Note** Installing or updating an operating system or kernal software to a shared root shuts down any clients booted from that server and reboots the server! If you are installing non-kernal applications to a shared root, the clients will not be shut down and the server will not be re-booted.

7. When the server has rebooted and finished the startup process, turn on the diskless clients. They will boot and configure the software.

If you are updating a large cluster, this process could take several hours.

The default operating system and application shared roots are created during the installation of the diskless server by the configure script of the diskless fileset. This will enable the creation of these shared roots with the proper attributes, including the description field that is to be used by HP's System Administration Manager (SAM) during its "Add a Client" operation. The configure script for the diskless fileset will use `swreg` to create empty shared roots and assign attributes to them. Private roots are created by SAM during its "Add a Client" operation.

## Changing Defaults and Options

You can change command behaviors and policy options by copying the default values found in the commented file */usr/lib/sw/sys.defaults*. You can edit these values and and place them in */var/adm/sw/defaults* or *$HOME/.sw/defaults*. You can also change default values interactively using the GUI/TUI Options Dialog.

---

**Note** Use caution when changing default option values. They allow useful flexibility but can produce harmful results if changed to a value that is inappropriate for your needs. Use the On-Line Help or consult the appendix Appendix A in this manual to understand fully each option and value.

---

Altering default values can help when you don't want to specify command behavior every time the command is invoked. Default values are specified in the defaults file using a `command.option=value` syntax.

These values can be individually overridden on the command line by specifying an alternative *optionfile* with the `-X` option or with the `-x` *option=value* option. For example, to start an interactive install session that lets you install software that is not compatible with the proposed host system, type

```
swinstall -i -x allow_incompatible=true
```

Defaults can also be edited directly in the */var/adm/sw/defaults* or *$HOME/.sw/defaults* files or changed using the GUI Options Editor.

Some of the defaults and options that apply to `swinstall` and `swcopy` are:

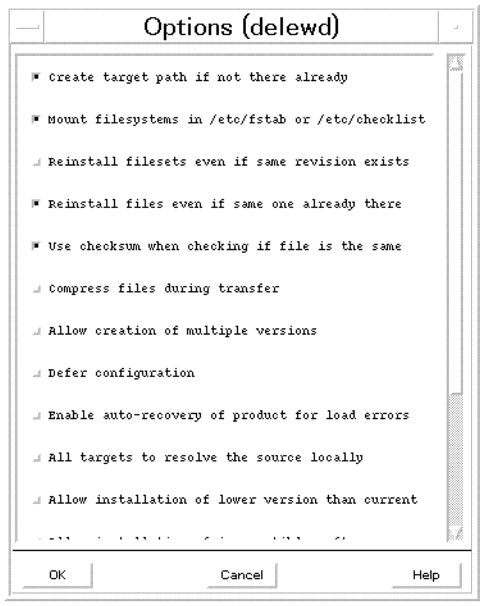
**Table 2-2. Install Defaults**

| | |
|---|---|
| allow_downdate=false | register_new_depot=true |
| allow_incompatible=false | register_new_root=true |
| allow_multiple_versions=false | reinstall_files=true |
| autoreboot=false | reinstall=false |
| autorecover_product=false | reinstall_files_use_cksum=true |
| autoselect_dependencies=true | retry_rpc=1 |
| codeword=xxxxx | rpc_binding_info=ncadg_ip_udp:[2121] |
| compress_files=false | rpc_timeout=7 |
| create_target_path=true | select_local=true |
| customer_id=xxxxx | software= |
| defer_configure=false | software_view=all_bundles |
| enforce_dependencies=true | source_directory=/var/spool/sw |
| enforce_dsa=true | source_tape=/dev/rmt/0m |
| enforce_kernbld_failure=true | source_type=directory |
| enforce_scripts=true | target_directory=/var/spool/sw |
| autoselect_reference_bundles=true | targets= |
| logdetail=false | uncompress_files=false |
| logfile=/var/adm/sw/<command>.log | use_alternate_source=false |
| loglevel=1 | verbose=1 |
| log_msgid=0 | write_remote_files=true |
| match_target=false | |
| mount_all_filesystems=true | |
| polling_interval=2 | |

You can change the "true/false" and other values (such as 0, 1, 5) to fit your specific requirements. One way to change the default values is to: choose the specific default from the commented listing in */usr/lib/sw/sys.defaults*; copy the default line to */var/adm/sw/defaults* or *$HOME/.sw/defaults*; then modify the value as you wish. The next time you invoke the command the system will use the new value. See Appendix A for a complete listing and explanation of all defaults.

If you want to change options from the GUI/TUI, use the Options Editor that is accessed from the `Options` -▶`Change Options` choice.

**Note** Using this Options Editor does NOT change the option permanently. The option is only changed for the duration of the session. To change options permanently, you must edit the defaults file.



Options (delewd)

■ Create target path if not there already

■ Mount filesystems in /etc/fstab or /etc/checklist

☐ Reinstall filesets even if same revision exists

■ Reinstall files even if same one already there

■ Use checksum when checking if file is the same

☐ Compress files during transfer

☐ Allow creation of multiple versions

☐ Defer configuration

☐ Enable auto-recovery of product for load errors

☐ All targets to resolve the source locally

☐ Allow installation of lower version than current

**Figure 2-6. Options Editor Dialog**

In this dialog, each default or policy option is shown in the left column of the Editor and to the right of each option is a "push button." The "True" or "False" values for each option are listed in a menu hidden underneath the push button. To select a value, push the button with the left mouse button and hold it down. A pop-up menu "replaces" the push button while the mouse is suppressed. While still holding down on the mouse button, move the cursor to the desired value. Release the mouse button. The pop-up menu disappears and the push button is re-labeled with the desired value.

The `OK` button may bring up a warning box if any of the options in the Options Editor have been modified and there are existing software selections, that is, if the Software Selection Window has been displayed previously and selections have already been made. If no options have been changed, selecting the `OK` button closes the Options Editor.

The `Cancel` button closes the Options Editor without applying any changes to the options.

For a description of each default and its values, place the focus on the default and press `f1`.

## Compressing Files to Increase Performance

SD-UX processes pass large amounts of data back and forth over the network, from depots to hosts, which might slow down network performance. The *compress_files* option can improve performance by first compressing files that are to be transferred. This can reduce network usage by 50% depending on the type of files. Binary files compress less than 50%, text files generally compress more. Improvements are best when transfers are across a slow network (approximately 50Kbytes/sec or less).

If set to "true," the *compress_files=* option will compress files, if not compressed previously by SD-UX, before transfer from a source. You may also choose the compression type and compression command to use when compressing and uncompressing files. See the `swinstall(1M)` man page for complete information on file compression.

This option should be set to "true" only when network bandwidth is clearly restricting total throughput. If it is not clear that this option will help, compare the throughput of a few install or copy tasks (both with and without compression) before changing this option value. See Appendix A for more information on compression defaults.

## Software Dependencies

Software that depends on other software to install or run correctly is considered to have a **dependency**. You can define how those dependencies are handled at installation. Dependency handling in `swinstall` and `swcopy` is affected by the *enforce_dependencies=* default, see Appendix A. In an upgrade to a new operating system version, it is strongly suggested that the *enforce_dependencies* default be set to "true."

Another default option that regulates dependencies is the *autoselect_dependencies=true* option. It determines if the system should automatically mark software for installation or copying based on whether it meets dependencies. Dependencies and their related software must BOTH be marked for installation or copying.

Software vendors can define two types of dependencies: **corequisites** and **prerequisites**. Dependencies can be specified between filesets within a product, including expressions of which revisions of the fileset meet the dependency. Dependencies can also be specified between a fileset and another product (the minimum subset of that product is a particular fileset within that product). Expressions for revisions and other product attributes are supported.

Corequisites

A corequisite relationship states that one fileset requires another to operate correctly, but DOES NOT IMPLY ANY LOAD ORDER.

| | |
|---|---|
| Prerequisites | A prerequisite relationship states that one fileset requires another to be installed and/or configured correctly before it can be installed or configured respectively. Prerequisites DO control the order of operations. |

If a dependency exists and there is more than one available object that resolves it, the latest compatible version that resolves it is automatically selected during the Selection Phase.

For a dependency to be resolved during `swinstall` or `swcopy` (if it is available from the source), it must be:

- complete (if the dependency is an entire product or subproduct it must completely exist in the source),
- in the proper software state on the source (that is, AVAILABLE) and
- free of errors (for example, no incompatibility errors).

If the dependency is NOT available from the source during `swinstall` or `swcopy` (or during `swverify` or `swconfig`), for SD-UX to resolve the dependency it must:

- exist on the target system,
- be complete (if the dependency is an entire product or subproduct it must be completely installed),
- be in the proper software state (the dependency must be CONFIGURED if the software dependent on it is to be installed and configured, INSTALLED if software dependent on it is to be installed but not configured, or AVAILABLE if the software dependent on it is to be copied) and
- be free of errors (for example, no incompatibility errors).

The *enforce_dependencies* option controls the behavior of the agent in dependencies.

To list the state of a dependency:

```
swlist -a state <fileset_name>
```
*or (for a depot)*
```
swlist -d -a state <fileset_name> @ /var/spool/sw
```

To list the complete contents of a product dependency:

```
swlist -a all_filesets <product_name>
```

To list the complete contents of a subproduct dependency:

```
swlist -a contents <product_name>
```

You can combine all these with multiple `-a` options (and the `-d` option if you are listing from a depot), see Chapter 5 for more information on listing attributes.

You can also find out what an object's dependencies are by pressing the `Dependencies ...` button in the Software Description Dialog (see "Getting More Information").

## Compatibility Filtering and Checking

`swinstall` normally filters out software products that are incompatible with the selected host. Compatibility means that the architecture of the hardware matches that required by the software (determined by the system's `uname` parameters). It also means that the OS version is the proper one for the software. The actual check for incompatible software is performed during the analysis phase. Compatibility filtering and checking are controlled by the *allow_incompatible* default option and depend on the host's `uname` attributes.

No compatibility filtering or checking is performed in `swcopy` because, just as your network may include many different host machines and architectures, your depot may also contain a wide variety of products designed for these varied computers or operating systems.

If *allow_incompatible=false*, then `swinstall` restricts the installation of incompatible software and automatically filters the products on the source. Only those products compatible with the hardware and operating system of ALL selected hosts are shown in the Object List. This is the default behavior.

The Message Area of the Software Selection Window contains the explanation:

```
Only software compatible with the target is available.
```

Note that when installing to alternate root file systems, compatibility filtering does not apply. HP-UX cannot assume the stand-alone system will be identical to the hardware and operating system of the alternate root system. It is up to you to know what will be compatible for the alternate root.

If *allow_incompatible=true*, `swinstall` allows the installation of any software. All products on the source are displayed for selection. The Message Area of the Software Selection Window contains the explanation:

```
All software on the source is available for selection.
```

**Note** HP strongly advises that you *do not* install software that is incompatible with the host.

## Installing Multiple Versions

Having multiple versions (and copies) of a software product installed at various hosts on the network is common because users may not all be using the same version or even running on the same machine platform. At least one copy or different version could exist on each host.

With `swcopy`, multiple versions of products are inherently possible in a depot. No special handling or checks are required.

With multiple version support, you can manage a diverse environment and even have multiple versions on a single host by installing relocatable versions into different product directories on the host. In addition to allowing different users to use different versions of the software, multiple version support also allows multiple copies of the same version in different locations (**locatable products**) if the software supports it. This may be needed if some users want significantly different configurations of the same software.

You can decide whether to allow multiple versions or not by controlling the *allow_multiple_versions* default option. If set to “false”, installed or configured multiple versions (that is, the same product, but a different revision, installed into a different location) are not allowed. While multiple *installed* versions of software are allowed, multiple *configured* versions are not recommended.

**Note** Managing multiple versions of a software product on your system requires close attention to the cross-product dependencies that may exist for each version. When installing multiple versions, make sure you also install multiple versions of the cross-product dependencies. If the dependencies are not relocatable and each version you want to install depends on a different version of the same product, multiple versions of the original product cannot be installed.

Once a multiple version is installed into a location, it is managed by specifying the “product” attribute (in contrast to specifying depot software by product “tags” or by other version attributes such as “revision” and “architecture”). This allows both old and new versions of software to be installed at the same time and even both configured if the software supports it. Multiple installed version supports both back-out of a defective version (by removing the new version and reconfiguring the old version, if necessary), as well as training and migration of users to the new version at their own pace.

Unauthorized, privately installed copies are avoided by controlling access to the IPD and restricting the use of the `swinstall` tool.

## Installing Software That Requires a System Reboot

Software packaged with the *is_reboot=true* attribute requires the host to be rebooted after the software is installed. However, when installing to alternate root file systems, the host will not be rebooted.

If a local installation is done that entails a reboot, the system reboots both the target and, more importantly, the controller, so there is no process left to report SUCCESS or FAILURE. SD-UX does not do a "reconnect" after reboot.

To find out if a software product requires the local host to be rebooted, get a description of the software either from the Software Selection Window using the menu item `Show Description of Software` or from the Analysis Dialog using the `Product Summary` and `Product Description` pushbuttons.

3

# Configuring and Verifying Software

The `swconfig` command lets you explicitly configure, unconfigure and reconfigure software products that are installed on a local host by executing the **configure script**. These scripts are only executed on the host that will actually be running the software. A fileset can also include an **unconfigure script** to "undo" a configuration. For more information on scripts, see Appendix B.

You can use the `swinstall` and `swremove` commands to perform many configuration or unconfiguration tasks. However, the `swconfig` command lets you work independently of these commands. For example, you can configure or unconfigure hosts that share software from another host where the software is actually installed. `swconfig` can also be useful when a configuration fails, is deferred, or must be changed.

## The `swconfig` Command

The `swconfig` command runs only from the command line interface. There is no Graphical or Terminal User Interface for `swconfig`.

The `swconfig` command provides the following features:

1. It configures the host on which the software will be run. A fileset can include a configure script to perform these actions on the host.

   The `swconfig` command also allows software to unconfigure the host on which it no longer will be run. That is, a fileset can include an unconfigure script that will undo the configuration done by the configure script.

   The configure and unconfigure scripts are usually run by `swinstall` and `swremove` respectively. They are not run when an alternate root directory is specified. Instead, the `swconfig` command must be used after that software has been made available to client hosts, to configure those hosts. Similarly, `swconfig` must be used on client hosts to unconfigure those hosts.

Automatic configuration can also be postponed on software installed to the root directory / (for example, when multiple versions are installed), by using the *defer_configure* default.

2. It only supports configuration of compatible software, controllable through the *allow_incompatible* default.
3. If a fileset relies on another software product for proper operation, that software product must be in a CONFIGURED state and is controlled by the *enforce_dependencies* option.
4. `swconfig` configures only one version of a fileset at a time, controllable through the *allow_multiple_versions* default.
5. By default, when `swinstall` installs software relative to the root directory /, it configures that software (controllable through the *defer_configuration* default). Software cannot be configured from an alternate root directory (that is, the alternate root directory must become some host's root directory before the software can be configured).
6. A vendor's configure script is as useful for those operations required for software updates as it is for new installs. The script must also be able to handle reinstallation and should attempt to design-in appropriate error control if data destruction is possible during downdates.
7. `swconfig` performs an analysis to see if the requested software exists, is in the proper state (INSTALLED) and prerequisites are (or can be) met.
8. `swconfig` moves software between INSTALLED and CONFIGURED states.

---

**Note** When the `swinstall` session includes a reboot fileset (such as when updating the core HP-UX operating system to a newer release), the configure scripts are run as part of the system startup processes after the system reboots.

---

## Configuring Software Using the Command Line

The syntax for `swconfig` is:

```
swconfig [-v] [-u] [-p] [-x option=value] [-f software file]
         [-t target file] [-X config file]
         [-C session file] [-S session file]
         [software selections] [@ target selections]
```

The `-t` *target file* option applies to the HP OpenView Software Distributor product.

## Sample Configurations

1. To configure productA, located in the default depot on the local host, you would type:

   ```
   swconfig productA
   ```

2. To unconfigure the software selections in the file *mysoft* that are installed in the default directory on the local host, you would type:

   ```
   swconfig -u -f mysoft
   ```

3. To reconfigure the Omniback product using the default option:

   ```
   swconfig -x reconfigure=true Omniback
   ```

4. To configure a particular version of Omniback:

   ```
   swconfig Omniback,r=2.0
   ```

   See "Command Line Software Selections" in Chapter 2 in Chapter 2 for information about use of the `,r=` version component.

## Command Options

The `swconfig` command supports the following options. (Except for the `-u` option, all `swconfig` options are a subset of those for `swinstall`.)

| **Option** | **Action** |
|---|---|
| `-u` | Causes `swconfig` to unconfigure the software instead of configuring it. |
| `-p` | Preview a configuration task from the command line by running it through the Analysis Phase and then exiting. You can use this option with any other options to understand the impact of a command before the system actually performs the task. |
| `-v` | Turn on verbose output to `stdout` and display all activity to the screen. Lets you see the results of the command as it executes. |
| `-x` *option=value* | Specify a value to override a default value or a value in an options file (see the `-X` *option file* option). See section |

|  |  |
|---|---|
|  | “Changing Options and Defaults” for more information on changing defaults. |
| `-f` *software file* | Read a list of software selections from a separate file instead of from the command line. In this software file, blank lines and lines beginning with # (comments) are ignored. Each software selection must be specified on a separate line. For an example of a software selection file, see “Command Line Software Selections” in Chapter 2 in Chapter 2. |
| `-t` *target file* | Specifies multiple *shared roots* on the local host. This option also applies to the HP OpenView Software Distributor product, which allows installtion to multiple remote hosts (targets). `-t` reads a list of these targets from a separate file instead of from the command line. |
| `-X` *option file* | Specifies a new option file. The default values for system options are provided in the file */var/adm/sw/defaults*. You can also provide a personal option file, *$HOME/.sw/defaults*. This option file overrides those values in the system defaults file. For a complete listing of system options, see the file */usr/lib/sw/sys.defaults*. This file lists the possible values and behaviors for each option for each command. See Appendix A for a complete listing and description of these defaults. |
| `-C` *session file* | Run the command and save the current option and operand values to a file for re-use in another session. |
| `-S` *session file* | Run the command based on values saved from a previous session. |

### Command Operands

The `swconfig` command supports the standard software selection (`bundle[.product[.subproduct][.fileset][,version]`) and target selection (`@ host[:/directory]`) operands. For an example of a software selection file, see “Command Line Software Selections” in Chapter 2 in Chapter 2.

## Changing Options and Defaults

In addition to the command-line option listed above, several command behaviors and policy options can also be changed by editing extended option and default values found in the defaults file */var/adm/sw/defaults*. Values in this file are specified using the *command.option=value* syntax. See Appendix A for a complete listing and description of these defaults.

## The Configuration Process

The configure process also has three phases:

- A Selection phase in which the local host resolves the list of software to configure
- An Analysis phase where the process goes through analysis to insure that the software selected can be configured successfully (existence, prerequisites, etc.)
- A Configure phase (the actual software configuration) where the configure or unconfigure scripts are executed and the software's state is changed between INSTALLED and CONFIGURED (or UNCONFIGURED).

### Selection Phase

For information on how to start a session, specify hosts, select software and specify products see Chapter 2 for complete information.

### Analysis Phase

This phase takes place on the local host. The analysis phase occurs before the loading of files begins and involves executing checks to determine whether the installation should be attempted. The load phase cannot be entered if there are any errors from the analysis phase.

**Note** No aspect of the local host's environment is changed (except where noted) during the analysis phase. Running the "preview" option (`-p`) on these tasks will not have a negative effect.

You can run the preview option to see if there are any errors or warnings. You can also return to the selection phase at this point and change selections. Errors in the analysis phase will only exclude those products that had errors in them. If only WARNINGS occur, the task will continue.

The sequential analysis tasks on the host are:

1. Initiate Analysis
2. Process Software Selections

   Get information from the Installed Product Database and check for compatibility.

   The system checks that all software is compatible with the host's `uname` attributes. This check is controlled by the default option *allow_incompatible*. If it is set to "false," the system produces an error; if set to "true," it produces a warning.
3. Check State of Versions Currently Installed

   If the product is "non-existent" or CORRUPT, the task issues an error that says the product cannot be configured and to use `swinstall` to install and configure this product.

   If the versions currently installed are not CONFIGURED and if the `-u` option is set (unconfiguring), the system issues a note that the selected file or fileset is already unconfigured.

   If the state of versions currently installed is CONFIGURED (if configuring), the check is affected by the reconfigure option. A note saying the fileset is already configured and will (*reconfigure=true*) or will not (*reconfigure=false*) be reconfigured is issued.
4. Check for Configuring a Second Version.

   This check is controlled by the *allow_multiple_versions* option. If it is set to "false," an error is generated stating that another version of this product is already configured and the fileset will not be configured. If it is set to "true," the second version will also be configured.
5. Check States of Dependencies Needed.

   An error or warning is issued if a dependency cannot be met. This is controlled by the *enforce_dependencies* option. If *enforce_dependencies* is set to "true" the fileset will not be configured. If *enforce_dependencies* is "false," the fileset will be configured anyway.

   If the dependency is a prerequisite, the configuration will likely fail.

   If the dependency is a corequisite, the configuration of this fileset will likely succeed, but the product may not be usable until its corequisite dependency is installed and configured.

## Configure Phase

The configure phase is entered once the selections have passed the analysis phase. This takes place on the local host.

The sequential relationship between the configure tasks is shown in the following list. Products are ordered by prerequisite dependencies if any. Fileset operations are also ordered by any prerequisites.

1. (Un)configure each product
2. Run each fileset's (un)configure script
3. Update the Installed Product Database to state (INSTALLED or) CONFIGURED

## Executing Configure Scripts

In this step, `swconfig` executes vendor-supplied configure or unconfigure scripts. The purpose of configuration is to configure the host for the software and configure the product for host specific information. For example, software may need to change the host's `.rc` setup, or the default environment set in */etc/profile*. Or you may need to ensure that proper codewords are in place for that host or do some compilations. Unconfiguration reverses these steps.

1. The scripts are executed, checking the return values.

   If an error occurs, the fileset is left in the state INSTALLED. If a warning occurs, the fileset will still be configured.
2. Scripts are executed in prerequisite order.

Configure scripts must also adhere to specific guidelines. For example, these scripts are only executed in the context of the host that the software will be running on, so they are not as restrictive as customize scripts. However, in a NFS diskless environment, they need to use file locking on any updates to shared files, as there may be multiple configure scripts operating at the same time on these shared files. Configure scripts cannot write to shared locations such as */usr* because they are read-only on client systems. Configure and unconfigure-configure scripts must be non-interactive.

For more information on scripts, see Appendix B.

## Verifying Your Installation

The SD-UX `swverify` command verifies available (copied), installed or configured software products on the specified host. It:

- determines whether installed or configured software is compatible with the host on which that software is installed.
- makes sure that all dependencies (prerequisites, corequisites) are being met (for installed software) or can be met (for copied software).
- executes vendor-specific verification scripts (that is, scripts that testify to the correctness of the product's configuration) if the installed state of the software is CONFIGURED.
- reports missing files, checks all file attributes including permissions, file types, size, checksum, mtime, link source and major/minor attributes.

### The `swverify` Command

The `swverify` command does not feature a Graphical User Interface. All verify interaction with the system is done on the command line.

The syntax for `swverify` is:

```
swverify [-d|-r] [-v] [-x option=value] [-f software file]
         [-t target file] [-X config file]
         [-C session file] [-S session file]
         [software selections] [@ target selections]
```

The `-t` option applies mainly to the HP OpenView Software Distributor product or when specifying multiple depots or roots in a target file.

#### Sample Verify Operations

1. To verify an installed fileset *mysoft.myfileset* located on the default depot at *myhosts*, you would type:

   ```
   swverify -d mysoft.myfileset @ myhosts
   ```

   You could also omit the @ sign and the *myhost* designation since the software being verified is assumed to be located in the default depot on the local host.
2. To verify the C and Pascal products that are installed on the local host:

   ```
   swverify C Pascal
   ```
3. To verify the HP Omniback product that is installed on the local host and watch the process (-v) on `stdout`:

```
    swverify -v Omniback
```

4. To verify the 2.0 version of Omniback that is installed on the local host at */opt/Omniback*:

```
    swverify Omniback,r=2.0 @ /opt/Omniback
```

## Command Options

The `swverify` command options are a subset of those for `swinstall` except the `-d` option, which verifies software on a depot instead of installed software.

| **Option** | **Action** |
|---|---|
| `-d` | Operate on a depot rather than installed software. |
| `-r` | (Optional) Operate on an alternate root rather than /. Verify scripts are not run. |
| `-v` | Turn on verbose output to `stdout` and display all activity to the screen. Lets you see the results of the command as it executes. |
| `-x` *option=value* | Specify a value to override a default value or a value in an options file (see the `-X` *option file* option). See section "Changing Options and Defaults" for more information on changing defaults. |
| `-f` *software file* | Read a list of software selections from a separate file instead of from the command line. In this software file, blank lines and lines beginning with # (comments) are ignored. Each software selection must be specified on a separate line. For an example of a software selection file, see "Command Line Software Selections" in Chapter 2 in Chapter 2. |
| `-t` *target file* | Specifies multiple *shared roots* on the local host. This option also applies to the HP OpenView Software Distributor product, which allows installtion to multiple remote hosts (targets). `-t` reads a list of these targets from a separate file instead of from the command line. |
| `-X` *option file* | Specifies a new option file. The default values for system options are provided in the file */var/adm/sw/defaults*. You can also provide a personal option file, *$HOME/.sw/defaults*. This option file overrides those values in the system defaults file. For a complete listing of system options, see the file */usr/lib/sw/sys.defaults*. This file lists the possible values |

| | |
|---|---|
| | and behaviors for each option for each command. See Appendix A for a complete listing and description of these defaults. |
| `-C` *session file* | Run the command and save the current option and operand values to a file for re-use in another session. |
| `-S` *session file* | Run the command based on values saved from a previous installation session. |

### Command Operands

As with `swinstall`, each software selection for a verify operation can be specified as a bundle, product, subproduct or fileset. If a bundle, product or subproduct is specified, all file sets in those objects will be included.

Local host selection can be specified by host name, host name alias or full network address optionally followed by an alternate directory.

If no directory is specified, the default root or depot directory on the local host (from the defaults file) is assumed.

### Changing Options and Defaults

In addition to the command line options listed above, several behaviors and policy options can be changed by editing default values found in the defaults file */var/adm/sw/defaults*. Values in this file are specified using the `command.option=value` syntax.

These values can be overridden by specifying an alternative options file with the `-X` option, or directly on the command line with the `-x` *option=value* option.

The following defaults are particular to `swverify`:

- *check_permissions=true*

  Verify owner, uid, group, gid, mode attributes of files and major/minor number for device files.

- *check_contents=true*

  Verify mtime, size and cksum of files.

- *check_scripts=true*

  Run the vendor supplied verify scripts when operating on installed software.

- *check_requisites=true*

Verify that the prerequisites and corequisites of filesets are being met.

- *check_volatile=false*

  Include installed files in the verification that have the *is_volatile* attribute set. By default, installed volatile files are not verified because their attributes may change in normal use.

See Appendix A for a complete listing and description of other defaults.

## The Verification Process

The software verification process has only two key phases: a selection phase and an analysis phase. For information on how to start a session, specify the host, select software and specify products see Chapter 2 for complete information.

### Verify Analysis Phase

The analysis phase for `swverify` takes place on the host. The host's environment is not modified.

The sequential analysis tasks on each host are:

1. Initiate Analysis
2. Process Software Selections

   The system accesses the Installed Products Database (IPD) or depot catalog to get the product information for the selected software.

   For installed software, the system checks that all products are compatible with its `uname` attributes. This check is controlled by the default option *allow_incompatible*.

   If *allow_incompatible* is set to "false", the system produces an error stating that the product is not compatible with the host.

   If *allow_incompatible* is set to "true", a WARNING is issued stating that the product is not compatible.

3. Check States of Versions

   The `swverify` command checks for correct states in the filesets (INSTALLED, CONFIGURED or AVAILABLE). For INSTALLED software, it also checks for multiple versions that are controlled by the *allow_multiple_versions* option.

If *allow_multiple_versions* is "false," an error is produced that multiple versions of the product exist and the option is disabled.

If *allow_multiple_versions* is "true," a WARNING is issued saying that multiple versions exist.

4. Check Dependencies An error or warning is issued if a dependency cannot be met. Dependencies are controlled by the *enforce_dependencies option*:

   If *enforce_dependencies* is "true," an error is generated telling you the type of dependency and what state the product is in.

   If *enforce_dependencies* is "false," a WARNING is issued with the same information.

   If the dependency is a corequisite, it must be present before the software will operate.

   If the dependency is a prerequisite, it must be present before the software can be installed or configured.

5. Execute Verify Scripts

   In this step, `swverify` executes vendor-supplied verify scripts only on installed software.

   a. A verify script is used to ensure that the configuration of the software is correct. Possible vendor-specific tasks for a verify script include:
   b. Determination of active or inactive state of the product.
   c. Check for corruption of product configuration files.
   d. Check for (in)correct configuration of the product into the OS platform, services or configuration files.
   e. Check licensing factors.
   f. Vendor-supplied scripts are executed and the return values generate an ERROR (if 1) or a WARNING (if 2).
   g. Scripts are executed in prerequisite order.

      For more information on scripts, see Appendix B.

6. File Level Checks

   File level checks are made with `swverify` for:

   a. Missing control_files, files and directories

b. Contents (mtime, size and checksum) for control_files and files
c. Permissions (owner, group, mode) for installed files
d. Proper symlink values

4

# Registering Software Depots

The SD-UX `swreg` command controls the visibility of **software depots** to those who are performing software management tasks. By default, the `swcopy` command registers newly created depots. Conversely, the `swremove` command unregisters a depot when it removes all software from it. You can invoke `swreg` to take explicit action in registering and unregistering depots when these automatic actions are not enough. For example, you would use `swreg` when:

- making a CD-ROM or other removable media available as a registered depot,
- registering a depot that is created directly with `swpackage`,
- unregistering a depot without physically removing it, thereby rendering it "invisible" to SD-UX commands.

See "Registering Depots Created by `swpackage`" in Chapter 11 for more information.

## Depot Commands

Just as with any other SD-UX command, `swreg` requires that you assemble options, targets and defaults into a command line or use command options to "point" to files containing targets, default behavior changes and other variables.

The `swreg` command uses the command line interface only. There is no Graphical User Interface for `swreg`.

The command syntax for `swreg` is:

```
swreg -l level [-u] [-v] [-C session file] [-S session file]
      [-t target file] objects_to_(un)register [@ host designation]
```

For example, to unregister a CD-ROM depot mounted at */mnt/cd*, type:

```
swreg -l depot -u @ /mnt/cd
```

Or, to register the same depot (mounted at */mnt/cd* on the local host) as a depot to be available on the network, type

```
swreg -l depot @ /mnt/cd
```

In either case, the `@` sign is optional on the local host.

## Command Options

The `swreg` command supports the following options:

| **Option** | **Action** |
|---|---|
| `-l` *level* (required) | List all objects at the specified *level*. Only one level may be specified. Supported levels are: |
| | `root` — Show the root level (that is, all the roots that are on the host). |
| | `shroot` — Show the shared roots on the local host. |
| | `prroot` — Show the private roots on the local host. |
| | `depot` — Show the depots on the local host. |
| `-u` | The "undo" option unregisters the depot. |
| `-v` | Causes verbose **logging** to stdout. |
| `-C` *session file* | Run `swreg` and save the current option and operand values to a file for re-use in another session. |
| `-S` *session file* | Runs `swreg` based on a previous session. |
| `-t` *target file* | Reads a list of targets from a file. The `-t` option is applicable for the HP OpenView Software Distributor product. |
| `-f` *object_file* | Read a list of *objects* to (un)register from a file. |
| `-X` *object_file* | Read a list of options and behaviors from a separate file. |
| `-x` *option=value* | Set a specific option value to *value*. Multiple `-x` options can be specified. |

### Command Operands

The `swreg` command operates on the host operand. This operand specifies the depot to be operated on by its absolute pathname (`host:path`, as with other commands).

## Changing Options and Defaults

In addition to the command-line options listed above, several behaviors and policy options can be changed by editing default values found in the defaults file `/var/adm/sw/defaults`.

See Appendix A for a complete listing and description of these defaults.

## Authorization

To register a new depot or to unregister-register an existing depot, `swreg` requires “read” permission on the depot in question and “write” permission on the host. To unregister a registered depot, the `swreg` command requires “write” permission on the depot. See Chapter 8 for more information on permissions.

## Registering Depots

The registration of depots (and roots) determines which hosts are available for software management tasks (that is, which hosts have a **daemon/agent** running). It enables the controller system to retrieve and present (via the GUI or `swlist`) a list of existing depots or roots from which a selection can be made. Registration information consists of the depot or root's identifier (its path in the host file system).

Note that registration does not imply access enforcement. Access enforcement is left to security (see Chapter 8).

# Part II
# Administering Software

5

# Listing Software

The `swlist` utility creates customizable listings of software products that are installed on your local host or stored in depots for later distribution.

With `swlist` you can:

1. Specify the "level" (bundles, products, subproducts, filesets or files) to show in your list.
2. Display a "table of contents" for a software source.
3. Specify a set of software **attributes** to display for each level. Software attributes are items of information about products contained in the Installed Products Database or in catalog files. These items can include the product's name or "tag," its size (in Kbytes), revision number, etc.
4. Display selected or all software attributes for each level.
5. Show the product structure of software selections.
6. List software stored in an alternate root directory.
7. Display the depots on a specified host.
8. Create a list of products, subproducts and/or filesets to use as input to the `swinstall` or `swremove` commands.

The SD-UX `swcluster` command also provides a listing capability. With its "list mode" (`swcluster -l`) you can display various parameters—shared and private roots, clients, booted_clients, bundles, products, subproducts and filesets—of an NFS diskless cluster. See "Installing/Updating Software on a Cluster Server" in Chapter 2 for more information on diskless clusters.

## The `swlist` Command

The `swlist` command does not feature a Graphical User Interface. All interaction with the system is done on the command line.

The `swlist` session consists of only the Selection Phase which takes place on the local host.

The syntax for `swlist` is:

```
swlist [-d] [-r] [-l level designation] [-v] [-R]
       [-x option=value] [-s source] [-f software file]
       [-t target file] [-X option file] [-a attribute]
       [-C session file] [-S session file]
       [software selections] [@ target selections]
```

On the command line, you specify which products to list (`-f` *software file* or *software selections*), what additional software information you want in the list (`-a` *attribute*), the level to which you want to list (`-l` *level designation*) and where the products are (on a local host, depot or directory). As with other commands, you can also specify sources, software and option changes.

The `-t` option is applicable to the HP OpenView Software Distributor product.

Remember, the @ character and the space following it, if used, are important to the proper operation of the command.

**Note** If you are piping the output of a `swlist` command to `more` (for example, `swlist -vl file |more` which would create a very large listing) and want to terminate the process, use `ctrl`-C. If you use `q` to quit, `swlist` may hang.

## A Simple List

To produce a list of the software (by name) installed at root (/) on your local host, you would simply type:

```
swlist
```

Which might produce a listing on your display like this:

```
# Initializing...
# Contacting target "xxyyzz"...
#
# Target:  xxyyzz:/

# Bundle(s):

B3782CA      B.10.10  HP-UX Media Kit (Reference Only. See Description)
B3898AA      B.10.10  HP C/ANSI C Developer's Bundle for HP-UX 10.0 (S700)
HPUXEngRT700 B.10.10  English HP-UX Run-time Environment

# Product(s) not contained in a Bundle:

HMS             1.01
OBAM3_0         B.10.00        ObAM 3.0
```

Using `swlist` with no options set and no software selected gives you a listing of all software bundles *plus all products that are not part of a bundle.* (Previous releases of `swlist` showed only bundles.)

By adding the `-d` option you would get the same listing of software residing in the default depot on your local host.

For a more thorough discussion of how to customize your lists using the `swlist` defaults and options, see the section "Advanced Topics for `swlist`" in the back of this chapter.

### Command Options

The `swlist` command supports the standard options as described in Chapter 2 plus these additional options:

| **Option** | **Action** |
|---|---|
| `-d` | List products available from a depot rather than software installed on the local host. |
| `-r` | (Optional) List products on an alternate root (instead of `/`). |
| `-l` *level designation* | |

List all software objects *down to* the specified "level designation." Level designations are the literals: depot, bundle, product, subproduct, fileset or file. You may use only one level designation per command. Do not use software names, subproduct names, etc. to specify levels.

The syntax is to choose a level as a starting point and list items only down to that level:

**Table 5-1. The `-l` Options**

| Option | Action |
|---|---|
| `swlist -l root` | shows the root level (roots on the specified target hosts) |
| `swlist -l shroot` | shows the shared roots |
| `swlist -l prroot` | shows the private roots |
| `swlist -l bundle` | shows only bundles |
| `swlist -l product` | shows only products |
| `swlist -l subproduct` | shows products and subproducts |
| `swlist -l fileset` | shows products, subproducts and filesets |
| `swlist -l file` | shows products, subproducts, filesets, files and numbers (used in software licensing). |

See the section "Advanced Topics for `swlist`" for more information on levels.

`-v` List all the attributes for the level given, one per line. These descriptive, state or relational attributes are displayed for each software selection (bundle, product, subproduct, fileset) or depot given. See the section "Verbose List" for a sample set of attributes.

If the `-v` option (verbose) is used in front of the `-l` option (that is, `-vl` *level designator*), all the attributes for each of the levels implied in the `-l` level designation specification are given. This lets you see all the attributes that are available for each software level and produces a lengthy list.

`-R` Shorthand for *-l bundle -l subproduct -l fileset.*

`-a` *attribute* Each listing level has its own set of attributes or information items. These attributes include such things as revision numbers,

**description**, vendor information, size in Kbytes and many others.

You may specify only one attribute per `-a` option. However, the `tag` attribute is always included by default, so specifying `-a` *revision* lists all product names AND their revision numbers.

For example, to list whether software bundles on a CD-ROM (mounted to the directory */SD_CDROM*) require a codeword or not, use the command:

```
swlist -d -a is_protected @ /SD_CDROM
```

See the section "Advanced Topics for `swlist`" for more information on attributes.

An attribute containing a large amount of information (for example, a README) is physically stored as a separate file and is displayed by itself if `-a` *README* is requested.

### Command Operands

In addition to the operands for the `-l` and `-a` options above, the `swlist` command provides the standard software selections and host selection operands as described in Chapter 2.

## Creating Software Lists

The starting point for a software list is always taken from the operands in the `-l` and `-a` options (or from the *level=* or *one_liner=* defaults, see the "Advanced Topics for `swlist`" section). You must decide what levels you want and what software attributes to list in addition to the product name.

**Note** Examples in this section were created with the *one-liner=* option left blank.

## Specifying Product Level

Specifying a level for a given software selection causes `swlist` to list the objects at that level plus all those that are *above* that level. Upper levels will be "commented" with a # sign. Therefore, only the level specified (product, subproduct, fileset or file) will be uncommented. This allows the output from `swlist` to be used as input to other commands. The exceptions are: 1) a list that contains only files; file-level output is not accepted by other commands and 2) a list that contains software attributes (`-a` and `-v`).

For example, if you wanted to see all the *products* installed on your local host, your command would be:

```
swlist -l product
```

and the listing would look like this:

```
NETWORKING
SAM
OPENVIEW
PRODUCT A
SOFTWARE Z
PRODUCT B
.
.
.
```

Note that the product names are uncommented because that was the level you requested to display and there are no levels above.

## Specifying Subproduct Level

For this example, on the local host, the NETWORKING product contains the subproducts ARPA and NFS and you want to see how big each object is (in Kbytes).

```
swlist -l subproduct -a size NETWORKING
```

```
# NETWORKING             9072
  NETWORKING.ARPA        4412
  NETWORKING.NFS         4660
```

The list does not show the files or filesets because you didn't specify that level on the command line.

If you wanted to see the names and revision numbers for the NETWORKING product on the local host, the command would be:

```
swlist -l subproduct -a revision NETWORKING
```

Remember, the product name is always assumed; you don't have to specify it in the `-a` option.

## Specifying Fileset Level

An example of using the `-l` option to generate a listing that includes all filesets for the product NETWORKING on the local host and a descriptive title for each:

```
swlist -l fileset -a title NETWORKING
```

```
# NETWORKING                         Network Software
# NETWORKING.ARPA                    ARPA/Berkeley Services
  NETWORKING.ARPA-INC           ARPA include files
  NETWORKING.ARPA-RUN           ARPA run-time commands
  NETWORKING.ARPA-MAN           ARPA manual pages
  NETWORKING.LANLINK            CORE ARPA software
# NETWORKING.NFS                     Network File System Services
  NETWORKING.NFS-INC             NFS include files
  NETWORKING.NFS-RUN             NFS run-time commands
  NETWORKING.NFS-MAN             NFS manual pages
```

Again, note the commented lines (#) representing the subproduct (NETWORKING.ARPA and NETWORKING.NFS) and product (NETWORKING) levels. The other lines are filesets.

If you are listing filesets on a depot (`swlist -l fileset -a title -d NETWORKING`), make sure the `-d` is in the proper position. The `-d` must PRECEDE the fileset name.

## Specifying Files Level

An example of the `-l` option to generate a comprehensive listing that includes all files for the subproduct NETWORKING.ARPA:

```
swlist -l file NETWORKING.ARPA
```

```
# NETWORKING.ARPA
# NETWORKING.ARPA_INC
  NETWORKING.ARPA_INC:/usr/include/arpa/ftp.h
  NETWORKING.ARPA_INC:/usr/include/arpa/telnet.h
  NETWORKING.ARPA_INC:/usr/include/arpa/tftp.h
  NETWORKING.ARPA_INC:/usr/include/protocols/rwhod.h
  NETWORKING.ARPA_INC:/usr/adm/sw/products/NETWORKING/ARPA-INC/index
  .
  .
  .

  # NETWORKING.ARPA_RUN
    NETWORKING.ARPA_RUN:/etc/freeze
    NETWORKING.ARPA_RUN:/etc/ftpd
    NETWORKING.ARPA_RUN:/etc/gated
    NETWORKING.ARPA_RUN:/etc/named

  .
  .
  .

  # NETWORKING.ARPA_MAN
    NETWORKING.ARPA_MAN:/usr/man/man8/ftpd
    NETWORKING.ARPA_MAN:/usr/man/man8/gated

  .
  .
  .
```

Note that the commented lines represent the requested level (NETWORKING.ARPA) plus one level up (fileset) from the specified file level (NETWORKING.ARPA_INC, NETWORKING.ARPA_RUN and NETWORKING.ARPA_RUN are all filesets). The uncommented lines are files.

## Depot Lists

Another class of objects that `swlist` can display are depot lists. This allows you to list all the registered depots residing on a local host. To do this, you can use a combination of the `-l` *depot* option:

**Table 5-2. Listing Depots**

| `swlist` **syntax** | **result** |
|---|---|
| `swlist -l depot` | list all depots on the local host |
| `swlist -l depot @ hostA` | list all depots on hostA |
| `swlist -l depot -v @ hostB` | list, in verbose mode, all depots on hostB |

## Verbose List

The -v option causes a verbose listing to be generated. A verbose listing is used to display all attributes for products, subproducts, filesets or files.

The verbose output lists each attribute with its name (keyword). The attributes are listed one per line. Given the length of this listing, you could post-process (filter) the output with grep and/or sed to see specific fields.

Attributes for a particular software level are displayed based on the software product name given with the swlist command. For example, swlist -v NETWORKING gives:

```
tag                 NETWORKING
instance_id         7869
control_directory
size                9072
revision      2.1
title               Network Software
mod_time
directory
vendor.information  Hewlett-Packard Company
is_locatable        true
architecture        HP-UX_9000/700
machine_type        9000/700
os_name             HP-UX
target.os_release   A.08*
```

If the -v option is used with the -l option, the cases are:

- To display all attributes for a bundle, use swlist -vl bundle.
- To display all attributes for a product, use swlist -vl product.
- To display all attributes for products and subproducts, use swlist -vl subproduct.
- To display all attributes for products, subproducts and filesets, use swlist -vl fileset.
- To display all attributes for products, subproducts, filesets and files, use swlist -vl file.

The table below provides a sample listing of the kinds of attributes that swlist will display. Not all these attributes exist for each software level or object. This list may change depending on vendor-supplied information. Do not use this list as the official list of all attributes. To get a complete list of the attributes for a particular level or object, use the swlist -vl *level designation* command (see above) or swlist -v *software name* (see example below) on your system.

**Table 5-3. Sample Attributes**

| Attribute | Description |
|---|---|
| architecture | Describes the target system(s) supported by the product |
| category | Type of software |
| copyright | Copyright information about the object |
| mod_time | Production time for a distribution media |
| description | Detailed descriptive information about the object |
| instance_id | Uniquely identifies this software product |
| title | Long/official name for the object |
| mode | Permission mode of the file |
| mtime | Last modification time for the file |
| owner | Owner of file (string) |
| path | Full pathname for the file |
| corequisite | A fileset that the current fileset needs (CONFIGURED) to be functional |
| prerequisite | A fileset that the current fileset needs to install or configure correctly |
| readme | Traditional readme-like information, release notes, etc. |
| revision | Revision number for an object |
| size | Size in bytes; reflects the size of all contained filesets |
| state | Current state of the fileset |

Here are some examples of verbose listings:

This command on the local host:

```
swlist -v NETWORKING.ARPA-RUN
```

produces this listing:

```
# NETWORKING.ARPA
fileset
tag            ARPA-RUN
instance_id    1
revision       1.2
title          ARPA run_time commands
size           556
state          configured
corequisite    NETWORKING.LANLINK
is_kernel      true
mod_time       733507112
```

This command:

```
swlist -vlfile NETWORKING.ARPA-RUN
```

produces this listing:

```
#NETWORKING.ARPA
tag:          ARPA-RUN
instance_id   1
revision      1.2
title         ARPA run_time commands
size          556
state         configured
corequisite   NETWORKING.LANLINK
is_kernel     true

file          etc/freeze
path          /etc/freeze
type          f
mode          0755
owner         bin
group         bin
uid           2
gid           2
mtime         721589735
size          24

file          etc/ftpd
path          /etc/ftpd
type          file
mode          0555
owner         bin
group         bin
uid           2
gid           2
mtime         721589793
size          9

.
.
.
```

## Advanced Topics for `swlist`

You can control the appearance and content of your lists by changing list default values in the defaults file `/var/adm/sw/defaults` or `$HOME/.sw/defaults`. Values in this file are specified using the `command.option=value` syntax. You can also change option values from the command line via the `-x` *option=value* option.

### List Defaults

Instead of repeatedly specifying the software levels and attributes on the command line each time you invoke `swlist`, you may use two special defaults: *level=* and *one_liner=*.

| | |
|---|---|
| *level=* | This default pre-determines what level to list: product, subproduct, fileset or file. For example, by setting this default to *level=fileset*, future `swlist` commands would always list everything down to, and including, filesets for each host, depot or product selected. |
| *one_liner= "attribute attribute attribute"* | Specifies the attributes (revision, size, title, etc.) the default listing should present. These attributes are separated by <tab> or <space> and enclosed in quotes (" "). Several attributes may be chosen but a particular attribute may not exist for all applicable software levels (product/subproduct/fileset). For example, the software attribute *title* is available for bundles, products, subproducts and filesets, but the attribute *architecture* is only available for products. |

In the absence of the `-v` or `-a` option in your command, `swlist` will always display the information as described in the *one_liner=* default for each software object level (bundle, products, subproducts and filesets), not for files.

### Creating Custom Lists

The `swlist` options and defaults allow you to create lists to fit your specific requirements. These lists can be as simple as listing the software products installed on your local host or as complex as a multiple column listing of files, filesets, subproducts, products and bundles installed.

For example, if you were to change the *one-liner=* option on the command line, the command:

```
   swlist -x one_liner="name revision size title"
```

produces this list of all the products installed on the local host:

```
   700RX           1.98    9845     700/RX X Terminal - all software
   ALLBASE         8.00.1  6745     Database Products
   C-LANG          2.5     5678     Programming Language
   DIAGNOSTICS     2.00    56870    Hardware Diagnostic Programs
   DTP68           2.00    26775    Desktop Publishing
   LISP-LANG       8.00.1  90786    LISP Programming Language
   WINDOWS         2.06    10423    Windowing Products
```

This listing shows, in columns from left to right, the product's tag, its revision number, its size in Kbytes and its title or full name.

**Note** Whatever you specify in the command line for software level and attributes will override the values in the default option files.

You can also change the *one_liner=* default value to {revision size title} in the defaults file. Then a listing of the C-LANG products on host2 would be:

```
   swlist C-LANG @ host2

   C-LANG.C-COMPILE     8.0    1346       C Compiler Components
   C-LANG.C-LIBS        8.0    2356       Runtime Libraries
   C-LANG.C-MAN         8.0    1976       Programming Reference
```

## More `swlist` Examples

In the following examples, `swlist` requests are sent to the standard output. All examples assume the *one_liner=* default is "revision size title" and the *level=* default is "product."

- List the contents of the local tape depot, /dev/rmt/0m:

```
   swlist -d @ /dev/rmt/0m
       or
   swlist -s /dev/rmt/0m

   AUDIT       3.5     9834      Trusted Systems Auditing Utils
   COMMANDS    1.7     4509      Core Command Set
   C-LANG      2.5     5678      C Programming Language
   DISKLESS    1.8     6745      HP Cluster Commands
   NETWORKING  2.1     9072      Network Software
   KERNEL      1.4     56908     Kernel Libraries and Headers
   VUE         1.3     5489      Vue (Instant Ignition Release)
   WINDOWS     2.06    10423     Windowing Products
```

- List all the media attributes of the local tape depot, /dev/rmt/0m:

  ```
  swlist -v -l depot @ /dev/rmt/0m
  ```

  *or*

  ```
  swlist -vl depot -s dev/rmt/0m
  ```

  ```
   type           distribution
   tag            CORE OS
   description    HP-UX Core Operating System Software Disc
   number         B2358-13601
   mod_date       June 1991
  ```

- List the README file for product, OS_CORE installed on the local host:

  ```
  swlist -a readme OS_CORE
  ```

  ```
  readme:
   ****************
   * Introduction *
   ****************

   The Release Notes for HP-UX Release X.0 contain an overview of the
   new/changed product features that are included in the release.  For
   detailed information about these features, refer to the appropriate
   product manuals.  This document does not contain information about
   software changes made as a result of a Service Request; that
   information may be found in the Software Release Bulletin (SRB) for
   Release X.0.

   ********************
   * Hardware Support *
   ********************

   The HP 9000 Model XXX is no longer supported.

  .
  .
  .
  ```

- List the products stored in the software depot on host1 located at */swmedia*. For this example assume the `swlist` *one_liner* is: "title size architecture":

  ```
  swlist -d @ host1:/swmedia
  ```

  ```
  FRAME      Frame Documentation Pkg     2319    HP-UX_9000_Series700/800_AorB
  FRAME      Frame Documentation Pkg     2458    OSF1_9000_Series700_1.0
  ME30       3-D Mechanical Eng          5698    HP-UX_9000_Series300/800_AorB
  SOFTBENCH  Softbench Development Env   4578    HP-UX_9000_Series300
  TEAMWORK   Teamwork Design & Analysis  3478    HP-UX_9000_Series300/400
  ```

  Note that the media contains two revisions of the FRAME product.

6

# Removing Software

The `swremove` command removes software that has been installed on a host. Before its removal, the software is first unconfigured (if it was installed in the default directory). `swremove` also removes software products that have been copied to a software depot but unconfigure scripts are not run on depot software (see "Removing Software from Depots").

The `swremove` command offers the following features:

- It removes files from the specified location. It removes symbolic links, but not the targets of symbolic links. It also lists busy files that were not removed.
- It supports the execution of control scripts (both product and fileset) including:
  - **checkremove scripts** that make sure the removal will succeed (if this check fails, the product cannot be removed). For example, the script could check to see if anyone is currently using the software.
  - **unconfigure scripts** that unconfigure the software on the host. For example, removing configuration from the host's `/etc/profile` or `/sbin/rc` files. This moves the software from the CONFIGURED state back to the INSTALLED state.
  - **preremove scripts** that perform actions before the removal such as moving a specific fileset to another location before removing the rest of the filesets in the product.
  - **postremove scripts** that perform actions after the removal such as replacing the above example fileset (that was moved to another location) back to its original location after removing the filesets.

  Preremove and postremove scripts cannot be used to unconfigure the host for the software or to unconfigure the software for the host. They are to be used for simple file management needs such as restoring files moved during installation.

- Before removal, `swremove` allows software to unconfigure the host on which it has been running. See the "Configuration/Unconfiguration" section in Chapter 3.

For more information on scripts, see Appendix B.

The SD-UX `swcluster` command is used to remove software from an NFS diskless server. In its "remove mode" (`swcluster -r`), `swcluster` supports removal of only non-kernel software. However, as long as at least one client is using a shared root, the kernel software cannot be removed (or else the client will cease to run. To completely remove a shared root, no client must be using it. For more information on removing software from diskless clusters, and "Installing/Updating Software on a Cluster Server" in Chapter 2

**Note** Removing a bundle *does not always remove all filesets in that bundle*. If a certain fileset is required by another bundle, that fileset will not be removed. For example, if the bundles `Pascal` and `FORTRAN` both use the fileset *Debugger.Run* and you try to remove `FORTRAN`, the fileset *Debugger.Run* will not be removed because it is also used by the bundle `Pascal`. This is done to prevent the removal of one bundle from inadvertently causing the removal of a fileset needed by another bundle.

## Removing Installed Software Using the Command Line

Just as with other SD-UX commands, to start a remove session via the command line you must assemble options, selections and defaults into a command line or use command options to point to files containing default changes and other variables.

### Remove Commands

The syntax for `swremove` is:

```
swremove [XToolkit Option] [-p] [-d|-r|-l] [-i] [-v] [-s source]
         [-x option=value] [-f software file] [-t target file]
         [-X option file] [-C session file] [-S session file]
         [software selections] [@ host selection]
```

## A Simple Removal

To remove a software product called MYSOFT from the default depot on the local host, type:

```
swremove -d MYSOFT
```

## Command Options

The `swremove` command supports the following standard options:

| **Option** | **Action** |
|---|---|
| `-p` | Previews the remove operation without actually removing anything. In the GUI/TUI, this option precludes advancing to the remove phase. You can only go through the Analysis phase. |
| `-d` | Operate on a depot rather than installed software. See the section "Removing Software From Depots" in this chapter for more information. |
| `-r` *alternate root* | (Optional) Specifies an alternate root directory. See "Advanced Topics for `swremove`" for more information on removing software from alternate roots. |
| `-l` | (Optional) Runs the command in **linkremove** mode, which removes the link between software installed in a shared root and the diskless client's private root. |
| `-i` | Runs a GUI or TUI interactive session. Used to "pre-specify" software selections for use in the GUI/TUI. |
| `-v` | Turns on verbose output. |
| `-x` *option=value* | Allows setting of an option on the command line. |
| `-f` *software file* | Reads the list of software selections from a file. |
| `-t` *target file* | Reads the list of target hosts from a file. This option applies to the HP OpenView Software Distributor product. |
| `-X` *option file* | Uses the options in the specified options file. |
| `-C` *session file* | Run the command and save the current option and operand values to a file for re-use in another session. |
| `-S` *session file* | Runs `swremove` based on a previous session. |

## Command Operands

The `swremove` command supports the standard software selection (`bundle[.product[.subproduct][.fileset][,version]]`) and host selection (`@ host[:/directory]`) operands.

See Chapter 2 for a more detailed discussion of Operands.

## Changing Options and Defaults

In addition to the command-line options listed above, several command behaviors and policy options can be changed by editing default values found in the defaults file `/var/adm/sw/defaults` or `$HOME/.sw/defaults`. Values in this file are specified using the `command.option=value` syntax. See "Advanced Topics for `swremove`" for more information on changing the values in these defaults and extended options.

## Removing Installed Software with the GUI/TUI

The `swremove` Graphical User Interface (GUI) and Terminal User Interface (TUI) are based on the `swinstall` and `swcopy` user interfaces that are described in detail in Chapter 2.

The `swremove` GUI and TUI also feature the same On-Line Help assistance as the `swinstall` interface. Choosing `Help` in the Menubar or using the `f1` key will present help topics that will further explain the various menu choices and options.

The `swremove` command behaves differently when removing from the primary root file systems versus alternate root file systems versus depots. The changes for alternate root file systems are noted in the section "Advanced Topics for `swremove`". The changes to the interface for removing from depots are summarized in the section "Removing Software From Depots" in this chapter.

The `swremove` session also consists of a Selection phase, an Analysis phase and a Remove phase. The Remove phase is the actual removal and monitoring of the process. The unconfigure scripts, preremove scripts and postremove scripts are executed in this phase.

`swremove` also automatically saves the current command options, source information, software selections and host/depots into a session file before the removal actually starts. This session file can then be recalled and re-executed. This session file is saved as *$HOME/.sw/sessions/swremove.last*.

To start the Graphical or Terminal User Interface for a remove session, simply type

```
/usr/sbin/swremove
```

and the Software Selection Window, displaying software on the local host, will appear.

Note that the `-r` and `-d` options must be specified for all root or depot removals, even when running the GUI/TUI.

You do not need to specify the `-i` option unless you want to pre-specify software to be marked in the GUI or TUI (see Chapter 2).

## Selecting Software to Remove

When you invoke `swremove` (without the `-i` or `-r` options), the Software Selection Window appears with all the software currently installed on the local host (at `/`) displayed in the Object List. Software is chosen by highlighting objects and "marking" them or by using the `Actions` -▸`Mark For Remove` menu choice. The Marked? flag is automatically updated to reflect the selection. The possible values for the Marked? flag are: "Yes", "Partial" and blank.

### Opening and Closing an Object

As in `swinstall`, the objects in the `swremove` Object List can be bundles, products, subproducts or filesets. The column headers are identical to the attributes in the `swinstall` interface. These product attributes can also be modified with the Columns Editor.

The objects in the lists may be opened to show their contents. For example, if you wish to see the subproducts of a particular product, you may open that product by double clicking on the item (GUI only) or moving the focus to that item and pressing `Return` (TUI only). The contents of the Object List will be replaced with a listing of the subproducts for that product. If you want to open the subproduct, you would double click on it and its filesets would be displayed. Objects that have an -▸ after the name can be opened to reveal other items.

To close an object in the GUI and return to the previous list, double click on the first item in the list (`. .(go up)`). In the TUI, move the focus to `. .(go up)` and press `Return`.

To open and close objects in the TUI, use the `Open Item` and `Close Level` menu choices.

When the product is opened, the subproducts and the filesets are shown in the list together. However, at the product level, only products are listed together.

## Analyzing a Removal

When the `Actions`→`Remove analysis` menu choice is made on the Software Selection Window, the Remove Analysis Dialog appears.

The Remove Analysis Dialog and its sub-menus are the same as those in the `swinstall` windows. See Chapter 2 for complete descriptions.

### Analysis Progress and Results

The attributes available in the Analysis Object List are identical to the attributes available in the `swinstall` Analysis Dialog.

Once analysis has completed, the Status should be "Ready." If any of the selected software can be removed, the status will be "Ready" or "Ready with Warnings." If none of the selected software can be removed, status will be "Excluded from task."

The Errors and Warnings are the same as those for `swinstall`. Refer to Chapter 2 for complete details.

The Products Ready column shows the number of products ready for removal out of all products selected. The total products ready includes those products that are:

- only marked because of dependencies
- marked inside of bundles
- partially and wholly marked.

See the section "Advanced Topics for `swremove`" for more information regarding dependencies.

A product may be automatically excluded from the removal if an error occurs with the product. The remove cannot proceed if the host target is excluded from the removal.

If the host fails the analysis, a Warning Dialog appears.

## Getting More Details

The Analysis Dialog presents more in-depth information regarding the host and the products being removed. There are three areas of additional information: the Product Summary, an overview of the products being removed, their revision numbers, the results of the attempt, the type of removal being attempted and a summary of any errors or warnings—the Product Description, a more complete presentation of the product, its size, vendor name, dependencies and other important information—the Logfile, a scrollable view of the information that is written in the logfile.

These dialogs can be displayed from the either the Remove Analysis Dialog or from the Remove Dialog by choosing the appropriate button.

Details for removals using the `-i` and `-r` options can only be displayed for one host at a time.

### Summarizing Details

The Product Summary describes the products to be removed and presents the effect of starting the removal on each product selected.

The Projected Action column describes what type of removal is being done. The possible types are:

- Remove

  The product exists and will be removed.
- Filesets Not Found

  The system did not find the filesets as specified.
- Skipped

  The product will not be removed.
- Excluded

  The product will not be removed because of some analysis phase errors. See the logfile for details about the error.

The Product Summary List is not an object-list; the products can not be opened and closed and actions can not be applied to the products. You cannot change the columns of the Product Summary List. To see more information about the products in the Product Summary, highlight an object in the list and select the `Product Description` button.

### Seeing the `swremove` Logfile

The `swremove` Logfile Dialog behaves exactly as the Logfile Listing in `swinstall`.

## The Remove Window

When the `OK` button choice is made in the Analysis Dialog, the Confirmation Dialog appears. The dialog explains that the remove phase will now be started on objects with a status of "Ready." To start the remove phase, press the `Yes` pushbutton on this dialog. (There are no "Final Remove Warnings").

The removal is then started on all objects with a status of "Ready" in the Remove Analysis Dialog. The Remove Analysis Dialog is replaced by the Remove Dialog.

The buttons are the same as those in the `swinstall` windows except there are no buttons for continuing, retrying unconfigure, aborting or rebooting. See Chapter 2 for complete descriptions.

## Monitoring the Removal Process

The columns available for the selected hosts in the Remove Dialog List are identical to those described in the `swinstall` Install Dialog. Unlike `swinstall`, however, you are not allowed to abort the removal UNLESS SD-UX suspends the task. Since removals can't suspend, the Abort and Resume buttons are not available.

When the remove phase is completed, a dialog is displayed to notify you of the completion. The only actions available at this point are:

- displaying the logfile,
- saving the session before exiting or starting with a new session,
- starting a new session,
- recalling a session, or
- exiting.

The Product Summary Dialog reflects the results of the removal on each product selected.

## Removing Software from Depots

When `swremove` is invoked with the `-d` option, software is removed from depots instead of root file systems. The user interface is slightly different when removing from depots.

A brief description of the `swremove -d` user interface is provided here. Where possible, you are referred to either the normal `swremove` or the Chapter 2 user interface descriptions.

### Selecting The Target Depot Path

Selecting depots for removal involves a depot path on the local host.
For example, on the command line, depots are specified with the syntax `local_host:<depot_path>`.

`swremove -d` can determine what depots currently exist on a host. When you invoke it, a Select Target Depot Path Dialog appears, superimposed over the Software Selection Window. It allows you to list all the depots registered on the Target Host (press the `Target Depot Path` button). You may choose a path from the list or enter one in the test box. When you have specified the depot path, the software in the depot will be listed in the Software Selection Window.

If no depots are currently registered on the local host, an error dialog appears with instructions to enter the name into the text area of this dialog.

### Selecting Software

The software selection menu items are identical to those available in this window during the normal `swremove`.

### Analyzing the Removal from a Depot

There are no changes in the Analysis Dialog.

### Removing the Product from a Depot

The only change to the Remove Window is that "Unconfiguring" is not a valid value for the Status attribute. There are no changes in the Remove Phase Details Window.

## Advanced Topics for `swremove`

The topics discussed in this section are for those users who have a firm understanding of the commands and who want to have additional control over their software removal process.

### Changing `swremove` Defaults

`swremove` default values can be changed in three ways: they can be individually overridden on the command line by specifying an alternative options file with the `-X` option or individually with the `-x` *option=value* option, they can be edited directly in the defaults file or you can use the GUI or TUI Options Editor Dialog to change them interactively. See Chapter 2 for more information.

### Removing Software from an Alternate Root

Software can be removed relative to the primary root directory (/) or relative to an alternate root directory. An alternate root is a non-root location that can function as the root of a stand-alone system; that is, a system that can be unmounted and function as a self-contained system. Any information files used in software removal are retrieved from the Installed Product Database (see "Installed Products Database" in Chapter 1) beneath this alternate root, not the IPD on the root volume.

The difference in behavior between `swremove` and `swinstall` is that `swremove` checks for the existence of the alternate root file system when it is specified. The GUI "Target Path Dialog" will appear first when `swremove -r` is invoked. It asks you to enter an alternate root path on the local host or choose one from the list of registered shared roots on the system. The default shared root that appears in the "Target Path Dialog" will be */export/shared_roots/OS_700*.

If the alternate root directory you choose does not exist (that is, is not registered), an error is generated and the dialog that prompted you for the path to the alternate root file system remains displayed.

### Removing Software From a Diskless Cluster

SD-UX provides the `swcluster` command to install, list and remove software from diskless servers and clients. Its `-r` option removes only non-kernel software from the cluster.

`swcluster` first contacts all booted clients in the cluster and unconfigures the software. It then removes the software from the clients and then from the server.

For example, if you wanted to remove the *UUCP* software product from a local HP Series 700 diskless cluster using very verbose (`-vvv`) output:

```
   swcluster -vvv -r UUCP @ /
```

## Dependencies

Dependents are enforced at Selection phase when the *autoselect_dependent* default is set to "true." This default determines if the system should automatically mark software for removal based on whether it depends on software you mark.

During Selection phase, if a dependent will be left on the system in an unusable state because of the removal of another software object, a dialog shows it. The software object that the dependent requires is not removed. This error can be overridden by setting the enforce_dependencies option in the defaults file to "false". A WARNING will still be generated, but `swremove` will show that it is being overridden.

When an object is selected (and *autoselect_dependent* is "true"), if there is more than one dependent available in the Software Selections Window, they are all automatically selected (not just the latest version of the dependent like `swinstall` dependency selection).

When removing from a depot, dependencies are handled the same as the normal `swremove`. Dependencies handling is controlled by the defaults *enforce_dependencies* and *autoselect_dependents*.

No compatibility filtering or checking is performed in any remove commands. There is nothing to be compatible with; software is already on the local host.

## Multiple Versions

Each separate version of a product along with its location directory is listed in the Object List. Selecting a multiple version implies a product:/location directory pair. By default, the location is not displayed in the Software Selection Window. It can be displayed using the Columns Editor `View`→`Columns ..` menu item. During the analysis phase, if the version of the product exists on the host but at a different location, a WARNING is generated.

More than one version of a product can be selected during the selection phase. During the analysis phase, if the product exists on the host, it will be removed. If it does not exist on the host, the product is simply skipped. The analysis Product Summary Window gives a product-by-product summary of what will be removed if the remove phase is started.

Multiple versions of products are inherently possible in a depot. No special handling or checks are required when removing from depots.

7

# Modifying IPD or Catalog File Contents

The SD-UX **Installed Products Database (IPD)** keeps track of installations, products and filesets on the system. Located in the directory */var/adm/sw/products*, the IPD is a series of files and subdirectories that contain information about all the products that are installed under the root directory (*/*). See Chapter 1 for more information on the IPD.

The equivalent IPD files for a depot are called **catalog files**. When a depot is created or modified using `swcopy`, catalog files are built (by default in */var/spool/sw/catalog*) that describe the depot and its contents.

Since both the IPD and catalog files are created and constantly modified by other SD-UX operations (`swinstall, swcopy` and `swremove`), they are *not directly accessible* if you want to change the information they contain. If you need to edit the information in either the IPD or in any depots' catalog files, you must use the SD-UX `swmodify` command.

## Using `swmodify` To Change/Add Software Information

The `swmodify` command adds, modifies, or deletes software objects and/or attributes defined in a software depot, primary root or alternate root. It is a direct interface with a depot's catalog files or a root's Installed Products Database. It does not change the files that make up the object, it only manipulates the information that *describes* the object.

Using `swmodify`, you can

- add new bundle, product, subproduct, fileset, control script or file definitions to existing objects
- remove software objects from a depot catalog file or root IPD
- change attribute values for any existing object (if you are adding a new object, you can define attributes for it at the same time).

## `swmodify` Examples

Here are some examples of how you can use `swmodify` to change catalog files or IPDs:

- Adding information.
  - To add files (*/tmp/a, /tmp/b. /tmp/c*) to an existing fileset:

    ```
    swmodify -x files=/tmp/a /tmp/b /tmp/c  PRODUCT.FILESET
    ```

  - If a control script adds new files to the installed file system, the script can use `swmodify` to make a record of the new files.
- Changing existing information.
  - To create some new bundle definitions for products in an existing depot:

    ```
    swmodify -d -s new_bundle_definitions \* @ /mfg/master_depot
    ```

  - If a product provides a more complex configuration process, a script can set the fileset's state to CONFIGURED upon successful completion.
  - To change the values of a fileset's attributes:

    ```
    swmodify -a state=installed PRODUCT.FILESET
    ```

  - To change the attributes of a depot:

    ```
    swmodify -a title=Master Depot -a
    description=&</tmp/mfg.description @ /mfg/master_depot
    ```

- Defining new objects.
  - You can import an existing application (not installed by SD-UX) by constructing a simple Product Specification File (PSF) describing the product and then invoke `swmodify` to load that definition into the IPD.
  - To create a new fileset definition (if the PSF contains file definitions, then add those files to the new fileset):

    ```
    swmodify -s new_fileset_definition
    ```

## The `swmodify` Command

The syntax for the `swmodify` command is:

```
swmodify [-d] [-r] [-p] [-v[v]] [-V] [-u] [-s product_specification_file]
        [-P pathname_file] [-a attribute=[value]]
  [-x option=value] [-X option_file] [-f file]
  [-C session file] [-S session_file] [software_selections]
```

Many of the options listed here for `swmodify` are the same for other SD-UX commands.

| Option | Action |
|---|---|
| `-d` | Perform modifications on a depot (not on a primary or alternate root). Your *target_selection* must be a depot. |
| `-r` | Perform modifications on an alternate root (and not the primary root). Your *target_selection* must be an alternate root. |
| `-p` | Previews a modify session without changing anything within the *target_selection*. |
| `-v[v]` | Turns on verbose output to `stdout`. (The `swmodify` logfile is not affected by this option.) A second `v` will turn on very verbose output. |
| `-V` | Lists all the SD *layout_versions* this command supports. |
| `-u` | If no `-a` *attribute=value* options are specified, then delete the specified *software_selections* from within your *target_selection*. This action deletes the definitions of the software objects from the depot catalog or Installed Products Database. If `-a` *attribute* options are specified, then delete them from within the given *target_selection*. |
| `-s` *product_specification_file* | The source Product Specification File (PSF) describes the product, subproduct, fileset, and/or file definitions that will be added or modified by `swmodify`. |

| | |
|---|---|
| | If a *product_specification_file* (PSF) is specified, `swmodify` selects the individual *software_selections* from the full set that is defined in the PSF. If no *software_selections* are specified, then `swmodify` will select *all* of the software defined in the PSF. The software selected from a PSF is then applied to the *target_selection*, with the selected software objects either added to, modified in, or deleted from it.<br><br>If a PSF is not specified, then *software_selections* must be specified. `swmodify` will select the *software_selections* from the software defined in the given (or default) *target_selection* . |
| `-P` *pathname_file* | Lets you specify a file containing the pathnames of files being added to or deleted from the IPD instead of having to specify them individually on the command line. |
| `-a` *attribute=value* | Add, change, or delete the *attribute value*. If the `-u` option is also specified, then it deletes the *attribute* from the given *software_selections* (or deletes the *value* from the set of values currently defined for the *attribute*). Otherwise, it adds/changes the attribute for each *software_selection* by setting it to the given *value*.<br><br>Multiple `-a` options can be specified. Each attribute modification will be applied to every *software_selection*.<br><br>The `-s` and `-a` options are also mutually exclusive: the `-s` option cannot be specified when the `-a` option is specified.<br><br>Note that you cannot change all attributes using the `-a` option. |
| `-x` *option=value* | Set the session *option* to *value* and override the default value (or a value in an alternate *options_file* specified with the `-X` option). Multiple `-x` options can be specified. |
| `-X` *option_file* | |

| | |
|---|---|
| | Read the session options and behaviors from *options_file*. |
| `-f` *file* | Read the list of *software_selections* from *file* instead of (or in addition to) the command line. |
| `-C` *session file* | Run the command and save the current option and operand values to a file for re-use in another session. |
| `-S` *session_file* | Execute `swmodify` based on the options and operands saved from a previous session, as defined in *session_file*. |

`swmodify` lets you specify a single, local *target_selection*. If you are operating on the primary root, you do not need to specify a *target_selection* because the target `/` is assumed.

When operating on a software depot, the *target_selection* specifies the path to that depot. If the `-d` option is specified and no *target_selection* is specified, then the default *depot_directory* is assumed.

## Operands

The `swmodify` command provides the standard software and host selection operands as described in Chapter 2.

## Changing Options and Defaults

In addition to the command-line options listed above, several command behaviors and policy options can be changed by editing default values found in the defaults file `/var/adm/sw/defaults`. Values in this file are specified using the `command.option=value` syntax. See Chapter 2 and Appendix A for more information on changing the values in these defaults and extended options.

The following keywords define source and target options as well as output and logging behavior that is specific to `swmodify`(all keywords are preceded with `swmodify.`):

| | |
|---|---|
| *source_file=* | Defines the default *product_specification_file* to read as the source. The `-s` option overrides this default. This default will only have effect if neither the `-s` nor the `-a` option is specified. |

|  |  |
|---|---|
| *target_directory=/var/spool/sw* | Defines the default distribution directory in which products will be modified. The *target_selection* operand overrides this default. This default only applies when the `-d` option is specified. |
| *verbose=0* | Controls the verbosity of the `swmodify` output (stdout). A value of 0 disables output to stdout. (Error and warning messages are always written to stderr). A value of 1 enables verbose messaging to stdout. A value of 2 enables very verbose messaging to stdout. |
| *loglevel=1* | Controls the log level for the events logged to the `swmodify` logfile. A value of 0 disables logging to the logfile; no information is logged. The default value of 1 enables verbose logging to the logfile. A value of 2 enables very verbose logging to the logfile. |
| *logdetail=false* | Controls the amount of detail sent to the logfile. A value of *true* allows additional detail depending on the loglevel specified. The default value of *false* restricts the amount of detail to POSIX (with *loglevel=1*) or POSIX and file level messages only (with *loglevel=2*). |
| *logfile=/var/adm/sw/swmodify.log* | Defines the default command logfile. |
| *files=* | When adding or deleting file objects, this option can list the pathnames of those file objects. There is no supplied default. (File objects being added or deleted can also be specified in the given *product_specification_file*.)<br><br>If there is more than one pathname, they must be separated by whitespace and surrounded by double-quotes. |
| *software=* | Defines the default *software_selections*. There is no supplied default. If there is |

| | |
|---|---|
| | more than one software selection, they must be separated by whitespace and surrounded by double-quotes. |
| *targets=* | Defines the default *target_selection*. There is no supplied default. |

## `swmodify` and the Product Specification File (PSF)

The *product_specification_file* (PSF) for `swmodify` uses the same `swpackage` PSF format as defined in Chapter 10.

A full PSF contains at least product, fileset, and file definitions. It can serve as valid input to the `swpackage` command, and provides `swmodify` with the information needed to create a new product, with filesets and files. A full PSF can also be specified when adding new filesets or files to existing products or filesets. A full PSF can also be specified when removing filesets (or files) from existing products (or filesets).

8

# Controlling Access to Software Objects

Along with the traditional HP-UX file access protection, all SD-UX hosts, depots, roots and products can also be protected from unauthorized access by special Access Control Lists (ACLs). These ACLs offer a greater degree of selectivity than do HP-UX permission bits.

## Access Control Lists

An ACL allows you to specify different access rights to many individuals and groups instead of just one of each.

**Note** With SD-UX, you can control ACLs only on your local host. If you need to modify ACLs on remote hosts, you must purchase the HP OpenView Software Distributor (HP Prod. No. B1996AA) which provides extended software management plus multi-site software distribution capabilities.

An ACL is a set of entries that are attached to a software object when it is created.

### ACL Entries

ACL entries define which users, groups and/or hosts have permission to access the objects. They consist of three fields (*entry_type*, *key* and *permissions*) separated by colons:

```
entry_type[:key]:permissions
```

For example, an ACL entry for an object might be:

```
user:fred:r-ctw
```

which means that a *user* named *fred* can `read`, `control`, `test` and `write` the object, but the dash signifies that he cannot `insert` or create new objects. Permissions (*c r w i t*) are explained later in this chapter. The order in which the permissions are specified is not critical.

The ACL *entry_type* must be one of these values:

**Table 8-1. ACL Entry Types**

| Type | Permissions Apply To |
|---|---|
| user | user **principal**, whose name is to be specified in the `key` field |
| group | group principal, whose name is specified in the `key` field |
| object_owner | the owner of the object |
| object_group | members of the group to which an object belongs |
| other | principals with no matching user and group entries |
| host | host systems (agents acting for users) |
| any_other | principals not matching any other entry |

The *user* and *group* of the object's owner are determined and automatically recorded at the time the object is created, based on the identity of the person who creates it. This information is recorded as *user*, *group* and *realm*. An *object_owner* or *object_group* entry type in an ACL causes the ACL Manager to look up the owner and group information on the object and, if a match to the requester is found, grant permissions as specified.

There may be many *user*, *group*, and *host* type entries per ACL, while there may be only one of each of the *object_owner*, *object_group* and *any_other* types. There may be at most one "local" (that is, no key) *other* entry and an unlimited number of "remote" (that is, keyed) *other* entries.

## ACL Keys

The second part of the ACL entry is the `key`. The table below lists the possible key values for specific entry types.

**Table 8-2. ACL Entry Key Values**

| Entry Type | Key Content |
|---|---|
| user | a user's name |
| group | a group name |
| other | |
| any_other | no key allowed |
| host | a host's name |

When listing the ACL, the host is printed in its Internet address form (for example, `15.12.89.10`) if the local system cannot resolve the address from its host lookup mechanism (DNS, NIS, or */etc/hosts*).

## ACL Permissions

There are five different permissions grantable by the ACL:

| | |
|---|---|
| control | Permission to edit or change the ACL. |
| test | Permission to test access to an object (that is, read the ACL) |
| insert | Permission to install a new product, depot or root. |
| write | Permission to change a host, depot, root or product. |
| read | Permission to list depot, roots and products and attributes. |

In the ACL entry, these permissions are abbreviated *c*, *t*, *i*, *w* and *r*. If you are granting all permissions, you may use the shorthand letter `a` instead of the `crwit` to denote "all" permissions. The meaning of permissions is different for different types of objects and the permissions do not have to appear in any specific order. Roots do not provide product level protection, so all permissions on products installed on roots are controlled by the ACL protecting the root itself. Product level protection is provided on depots in this way: the depot's ACL protects the depot itself while product ACLs protect the products within the depot.

The table below summarizes object permissions and ACLs to which they may be applied.

**Table 8-3. ACL Permission Definitions**

| Permission | Allows You To: | | | |
|---|---|---|---|---|
| | **Host system** | **Root** | **Depot** | **Product on depot** |
| [c]ontrol | Modify ACLs | Modify ACLs | Modify ACLs | Modify ACLs |
| [t]est | Test access to an object, read (list) the ACL itself | | | |
| [i]nsert | Insert a new depot or root | Insert a new product | Insert a new product | N/A |
| [w]rite[1] | change host | chang e root or products | change depot | change product |
| [r]ead[2] | list depots and roots | list root & prod attrs | list depot & prod attrs | read product files |

1 Write permission means permission to change or delete the object, except the host source object may not be deleted.

2 Read permission on containers (i.e. hosts, roots and depots) is permission to list the contents; on products it is permission to copy/install the product.

## Editing ACLs

You can view or change ACL entries and permissions by using the `swacl` command, an SD-UX tool that allows you to list and change ACLs.

The syntax for `swacl` is:

```
swacl [-v] -l level
      [-M acl_entry|-D acl_entry|-F acl_file]
      [-x option=value] [-f software file] [-t target file]
      [-X option file] [full_object_name]  [@ targets]
```

The `-t` option applies only to the HP OpenView Software Distributor product.

### Command Options

In addition to the standard options (`-x`, `-f`, and `-X`, see Chapter 2), the `swacl` command supports these options:

`-v` Turn on verbose output to stdout. Lets you see the results of the activity while it is being performed.

`-l` *level* Level to edit. Level designations are the literals: host, depot, root, product, product_template, global_soc_template or global_product_template. ACL templates are discussed in the section "ACL Templates" in this chapter.

You can change an ACL with of any of the following options (if none are used, `swacl` just prints the specified ACLs). These options are mutually exclusive.

| | |
|---|---|
| `-M` *acl_entry* | Adds a new ACL entry or changes the permissions of an existing entry. You can enter multiple `-M` options. |
| `-D` *acl_entry* | Deletes an existing entry from the ACL associated with the specified object. You can enter multiple `-D` options. |
| `-F` *acl_file* | Assigns the ACL information contained in *acl_file* to the object. All existing entries are removed and replaced by the entries in the file. You can enter only one `-F` option. |

A typical *acl_file* looks like this:

```
# swacl    Installed Software Access Control List
#
# For host:  prewd:/
#
# Date:  Wed May 19 16:39:58 1993
#
# Object Ownership: User=root
                    Group=sys
                    Realm=prewd.fc.hp.com
# default_realm=prewd.fc.hp.com
object_owner:crwit
user:rml:crwit
user:root@newdist.fc.hp.com:crwit
group:swadm:crwit
any_other:crwit
```

The header information (lines marked with #) gives the object's name and owner and the name of the user's **realm** or hostname of his/her system.

## How `swacl` Works

The `swacl` command, when invoked without the `-M, -D` or `-F` options, reads the specified ACL, converts it into plain text and prints it to the screen so you can see it. If you re-direct the output of the command to a file, you can then edit that file and change the permissions in it. After editing, you can use the `-F` file option described above to replace the old ACL. This procedure gives you full ACL editing capabilities.

You must have "test" permission within the ACL to produce the edit file (list the ACL) and "control" permission to change it with `-F, -D` or `-M` options.

If the replacement ACL contains no detectable errors and you have the proper permission on the ACL, the replacement succeeds. If the replacement fails because you lack permission to make the change, an error is generated and the object is skipped.

You may change or delete existing entries, or you may add additional entries to the ACL.

---

**Caution** It is possible to edit an ACL so that you cannot access it! Caution should be used to avoid accidentally removing your own control *(c)* permissions on an ACL. As a safeguard, the local super-user should always edit ACLs.

---

## How ACLs are Matched to the User

It is important to note that permissions in ACLs are determined by a match to a single ACL entry (except for group entries), not to an accumulation of matching entries. Simply put, checking is done from the most restrictive entry types to the broadest. If a match is found in a user entry type, no further checking is done, and the permissions for that user are fully defined by the permissions field of the matched entry. That the matched user may be a member of a group with broader permissions is of no consequence.

---

**Note** The local host super-user has access to all local objects, irrespective of ACLs.

---

The ACL matching algorithm is:

1. If user is local superuser, then grant all permissions, else
2. If user is owner of the object, then grant “object_owner” permissions, else
3. If user matches a “user” entry, then grant “user” permissions, else
4. If any “group” entries match, then accumulate the permissions granted by all group_entries that match the user’s primary and supplementary groups, else
5. If an appropriate “other” entry matches, then grant “other” permissions, else
6. If an “any_other” entry, then grant “any_other” permissions, else
7. Grant no permissions

## How ACLs Protect Objects

The control of product insert/delete permissions differ on roots and depots.

The permission for anyone to insert or delete a product on a root is contained within the root's ACL. If you have write permission on a root, you can change or delete any product on that root; there is no product level control on roots.

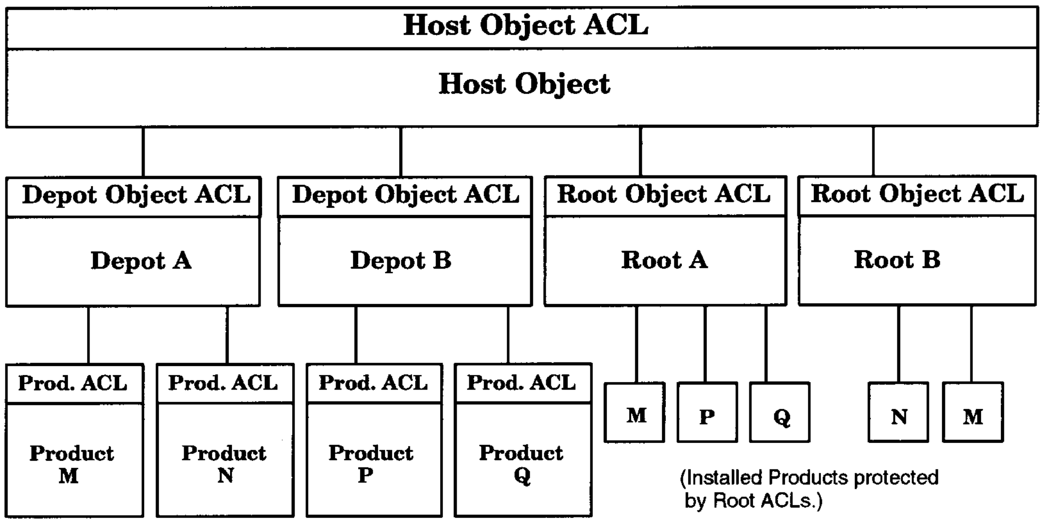


**Figure 8-1. Access Control Lists**

The depot ACL controls insertion (that is, creation) of new products, while the inserted object has its own ACL controlling modification and deletion. This allows the creator (that is, the owner) of a product on a depot to make changes to, or even delete, the product without requiring the broader write permission that could affect other users' products on the same depot.

This is useful for product control, because it allows you to assign management control for a specific product, once it is created, to a delegated administrator. Also, when a product is created on a depot, the user and group identity of the creator is recorded in the product information.

If the product ACL contains an *object_owner* entry granting write permissions to the owner, then the product creator will automatically have rights to change or delete the product. Therefore, the depot can be more widely opened to insertion because users with insert permission can only copy in new products or delete their own products: you don't have to worry about a user erroneously (or maliciously) deleting some critical product that they shouldn't control.

The rationale for this protection scheme is borrowed from a mechanism introduced in the BSD file system. With "write" permissions on a BSD directory, you may create a file in the directory. If the sticky mode bit is set on the directory, only the file owner, the directory owner or superuser may remove or rename the file.

For example: In */tmp*, owned by root, with wide-open "write" permission and the sticky bit set (that is, mode 1777), anyone can create files that nobody else (except themselves and superuser) can remove, making */tmp* a more secure place to store temporary work because someone else can't delete your files there.

Installing or copying from an unregistered depot requires you to have insert permission on the depot's host. If this permission is denied, the depot's daemon log will contain the message:

```
ERROR:   Access denied to SD agent at host lucille on behalf of
         rob@lucille to start agent on unregistered depot
         "/users/rob/depot".  No (i)nsert permission on host.  07/23/93
         15:51:06 MDT
```

This message shows that it is the AGENT at *lucille* that did not have insert permission on the depot's host, not the user *rob@lucille*.

The remote host ACL must have two entries granting insert permission: one for the user, and one for the target host.

For example, for user "rob" to be allowed to install a product on his host "lucille" from an UNREGISTERED depot on source host "desi", the command

```
swacl -l host @ desi
```

must show the minimum ACL entries:

```
user:rob@lucille:---i-
host:lucille:---i-
```

Note that "rob" could alternatively register the depot with the `swreg` command with only the first entry above before running `swinstall`or `swcopy`.

### Host System ACLs

The host system is the highest level of protected object. An ACL protects each host, controlling permission to create depots and roots on the host. A **host ACL** may grant the following permissions:

| | |
|---|---|
| read (r) | Permission to obtain host attributes, including a list of depots and roots on the host. |
| write (w) | Permission to change the host object. |
| insert (i) | Permission to create and register a new depot or root on the host. |
| control (c) | Permission to edit or change the ACL. |
| test (t) | Permission to test access to an object and list the ACL. |

A sample host system ACL that grants depot and root source creation, source listing and ACL administration to a user named "rob" and give open permission to list the depots and roots on the host, would be:

```
user:rob:r-ic-
any_other:r----
```

Note that since *any_other* does not have (t)est permission, only rob can list this ACL, because he has (c)ontrol permission.

### Root ACLs

Principals (users) identified in ACLs that are protecting roots are granted permission to manage **installed products**. The permissions associated with a root are:

| | |
|---|---|
| insert (i) | Permission to install a new product. |
| read (r) | Permission to list the contents of the root. |
| write (w) | Permission to delete the root itself or the products in the root. |
| control/test (c/t) | Same as host ACLs above. |

A sample root ACL that grants a user named "lois" permission to read, write and insert software and members of the group named "swadm" all possible permissions is:

```
user:lois:--rwi-
group:swadm:crwit
```

When a root is created, it is automatically protected by a default ACL derived from its host. `swacl` can be used to change the initial values of this ACL. See the section on "ACL Templates."

## Depot ACLs

Principals identified in ACLs that are protecting depots are users who have been granted permission to manage the depot and to create new products. The permissions associated with a depot are:

| | |
|---|---|
| insert (i) | Permission to copy a new product into the depot. |
| read (r) | Permission to list the contents (products) of the depot source. |
| write (w) | Permission to delete the depot (if it is empty), and unregister itself (not the products in the depot). |
| control (c) | Same as above. |
| test (t) | Same as above. |

A sample depot ACL that grants

1. its creator all permissions;
2. user "george" permission to list and insert software products;
3. members of group "swadm" permission to list and insert products, change the ACL and delete the depot itself; and
4. everyone else permission to list the contents of the depot,

would be:

```
object_owner:crwit
user:george:-r-i-
group:swadm:crwi-
any_other:-r---
```

When a depot is created, it is automatically protected by a default ACL derived from its host. Products inserted in that depot will automatically be protected by an ACL derived from the depot. This concept is discussed in the section "ACL Templates."

## Product ACLs

Product ACLs only apply to products on depots. Products on roots are protected by the root's ACL. There are two classes of **principal** that are granted access rights to products:

| | |
|---|---|
| users | Granted various administrative permissions. This class includes groups and others, both local and remote. |
| hosts | Target systems (agent/daemons) granted read permissions to allow product installation. |

Permissions on products are:

| | |
|---|---|
| write (w) | Permission to users to change and delete the product and/or product information. |
| read (r) | Permission granted to target_hosts to read the source-depot product. (that is, grant permission to a remote system to install the protected product). |
| control (c) | Permission to edit or change the ACL. |
| test (t) | Permission to test access to an object. |

A sample product ACL that grants user “swadm” and the creator of the product all permissions and allows open read permission (allowing free distribution to all systems) would be:

```
user:swadm:crw--
object_owner:crw--
any_other:-r---
```

Note again, that when a product object is created, it is automatically protected by a default ACL from the depot/root source or, absent that, one from the host.

## ACL Templates

There are two special ACLs that are used to create initial ACLs to protect newly created objects - **product ACL templates** (*global_product_templates* or *product_templates*) and **container ACL templates** (*global_soc_templates*).

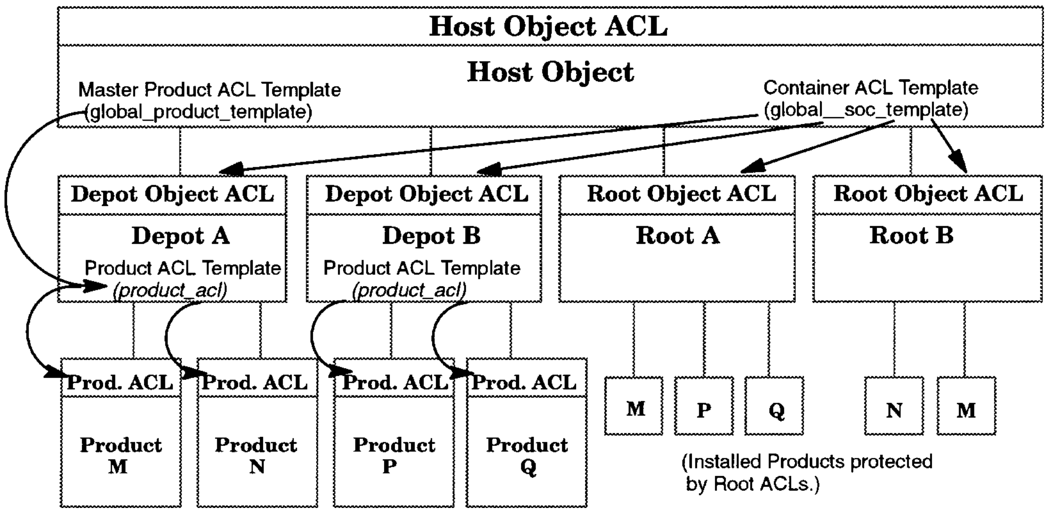


**Figure 8-2. ACL Templates**

When a product is installed on a depot with `swcopy` or `swpackage`, the system uses a product ACL template (provided by the depot that contains that product) to define the initial permissions of the new product's ACL.

The system uses the product ACL template of the host system (*global_product_template*) to initialize the product ACL template of the new depot and uses the container ACL template of the host system (*global_soc_template*) to initialize depot and root ACLs.

Thus, there are three ACLs on the host:

- Host ACL

  Attached to, and controlling access to the host object itself.
- Container ACL Template (global_soc_template)

  Used to initialize the ACL protecting new depots and roots created on the host.
- Product ACL Template (global_product_template)

  The ACL that is used to initialize the product ACL template on depots that are created on the host.

There are also two ACLs on product depots:

- The depot's ACL that is used to determine permissions on the depot.

- The depot's product ACL template (*product_template*) that is used to initialize the ACLs protecting new products on the depot.

There is one ACL on the installation (root):

- The root ACL that protects the root and products installed on it.

And finally, there is one ACL on the product:

- The product's ACL that is used to determine permissions on the product.

Every host must have an ACL protecting it and a pair of template ACLs (product and container) to provide initialization data for implicit depot and product ACLs. All three are created when HP-UX is installed on the host.

## Default ACL Template Entries

The host system's container ACL template dictates initial permissions on all depots and roots that are introduced on that host. The host also contains a master copy of a product ACL template that is copied to each new depot. A default set of host ACLs is provided at the time HP-UX is installed that can be altered by the system administrator. The contents of these host-system ACLs immediately after installation are:

### Host ACL

The Host ACL below grants permission to group "swadm" to insert and delete depots and roots on the host and to change this ACL. It also allows global (*any_other*) permission to list the depots and roots on the host and to list the ACL itself:

```
object_owner:swadm:crwit
group:swadm:crwit
any_other:-r--t
```

Remember, the local superuser always has all permissions, even without an ACL entry.

To take advantage of "swadm" entries in these examples, the "swadm" group must be created by the system administrator.

### Container ACL Template

The container ACL template below grants the owner or creator (*object_owner*) of a new depot or root permission to manage that new depot or root and to change its ACL. It also grants global permission (*any_other*) to list products in the new depot or root.

```
object_owner:crwit
group:swadm:crwit
any_other:-r--t
```

**Product ACL Template**

The product ACL template below grants permission to perform all operations on products installed on depots on this host to the respective creator (that is, *owner*), via the *object_owner* entry, of each product. It also grants permission to read (that is, install) and test the product to "any host" (the *any_other* entry).

```
object_owner:crwit
group:swadm:crwit
any_other:-r--t
```

In addition to encompassing all hosts, the *any_other* entry also applies to all other users except, in this case, the product's owner. However, product read permission has meaning only to host principals, and other possible product permissions never apply to hosts, therefore the *any_other* entry may be overloaded with user and host permissions, if desired, without any danger of ambiguity. This overloading should be kept in mind when using the SD-UX commands to execute solutions.

These host ACL defaults provide a good starting point for control over the management functions while providing open access to read the software for installation on root targets.

## Task-Specific Permission Requirements

### Packaging

- If the depot does not exist, `swpackage` verifies that the user has insert permission on the host.
- `swpackage` verifies that the user has insert permission on a depot.
- `swpackage` verifies that the user has write permission on a product, if it already exists.

## Listing

- To list potential depots, the source agent verifies that the user has read permission on host.
- To list potential products, the source agent verifies that the user has read permission on depot or root.

## Copying

- Any list operations required to facilitate this function must be checked as described in the `swlist` section above.
- If the depot does not exist, `swcopy` verifies that the user has insert permission on the target host.
- The agent verifies that the user has insert permission on the depot.
- The agent verifies that the controller user has write permission on the product, if it already exists.
- The source agent verifies that the system has read permission on the source product.
- The source (depot) agent verifies that the depot is registered. If not, the agent verifies that the controller user and the target agent system each has insert permission on the source's host.

## Installing

- Any list operations required to facilitate this function must be checked as described in the `swlist` section above.
- The agent verifies that the user has insert permission on the target root.
- The agent verifies that the user has write permission on the root, if the product already exists.
- The source (depot) agent verifies that the system has read permission on the source product.
- The source (depot) agent verifies that the depot is registered. If not, the agent verifies that the controller user and the target agent system each has insert permission on the source's host.

## Removal

- If the object is a product on a depot, the agent verifies that the user has write permission on the product.
- If the object is a product on a root, the agent verifies that the user has write permission on the root.
- If the object is a depot or a root, or the last product contained in one of these, before removing the container the agent must verify that the user has delete permission on the root or depot.

## Configuration

The same permission checks are made as for the `swremove` operation above, except that this command does not apply to depots.

## Verify

- If the object is a product on a depot, the agent verifies that the user has read permission on the product.
- If the object is a product on a root, the agent verifies that the user has read permission on the root.

## Registering Depots

- To register a new depot, `swreg` requires "read" permission on the depot in question and "insert" permission on the host.
- To unregister a registered depot, the `swreg` command requires "write" permission on the host.

## Sample ACLs for Editing

Here are some examples based on the following ACL that is protecting a product (FORTRAN) created by user rob whose local host is `lehi.fc.hp.com`:

```
# swacl    Product Access Control Lists
#
# For host:  lehi:/
#
# Date:  Wed May 19 16:39:58 1993
#
# For product: FORTRAN,r=9.0,v=HP
# Object Ownership:   User=root
                      Group=sys
                      Realm=lehi.fc.hp.com
# default_realm=lehi.fc.hp.com
object_owner:crwit
user:barb:-r--t
user:ramon:-r--t
group:swadm:crwit
host:alma.fc.hp.com:-r--t
any_other:-r--t
```

To list the ACL for the product FORTRAN in depot /var/spool/sw (the default depot) and prepare it for editing, type:

```
swacl -l product FORTRAN &>acl_tmp
```

which will bring the above ACL into the file acl_tmp, ready for editing. Edit the acl_tmp file with any suitable text editor.

To replace the modified ACL, type:

```
swacl -l product -F acl_tmp FORTRAN
```

To edit the default product template on a depot /var/spool/sw_dev, use:

```
swacl -l product_template @ /var/spool/sw_dev $>tmp_file
```

then edit the tmp_file and replace the ACL:

```
swacl -l product_template -F tmp_file @ /var/spool/sw_dev
```

To delete user barb's and group swadm's entries:

```
swacl -D user:barb  -D group:swadm -l product FORTRAN
```

To give user ramon permission to change the product FORTRAN:

```
swacl -M user:ramon:trw -l product FORTRAN
```

To add an entry for user pam with complete management permission:

```
swacl -M user:pam:a        ["a" is shorthand for "crwit"]
```

To add an entry to grant every user in group swadm at remote hosts dewd and stewd full management control of the product FORTRAN on the default local depot:

```
swacl -M group:swadm@dewd:a -M group:swadm@stewd:a -l product FORTRAN
```

To list the ACL protecting the default depot at host dewd:

```
swacl -l depot @ dewd:
```

# Part III
# Creating Software Packages

- Packaging Overview
- Packaging Specification Files
- Analyzing and Building Packages

9

# Packaging Overview

This chapter describes the key points that software vendors and system administrators need to know about packaging software and data files into the proper format for distribution and installation with the SD-UX software management commands.

## Packaging Concepts

The software objects that SD-UX packages, distributes, installs and manages are files. A "packager" uses these files after they have been built (compiled) and installed into specific directory locations by the software "build" process. These directory locations range from separate, unconnected directory trees to the specific file locations that are required to make the software run on your system. You can specify files by a root directory (gathering all files below it) or by individual files. The file attributes can be taken from the files themselves, specified separately for each file or specified for a set of files.

The SD-UX packager provides a flexible packaging specification that fits into many software build and manufacturing processes.

### Determining Product Contents

The first step in packaging software is to determine what files and directories you want included in your software or data. These files and directories must follow certain guidelines so that the desired configurations can be supported.

Key points in this structure are: where are shareable (for example, executables) and non-shareable (for example, configuration) files installed, and how is configuration used to put non-shareable files in place?

## Vendor- or Packager-Supplied Attributes

SD-UX uses a special "Product Specification File" (PSF) to describe the physical make-up of a product package. The PSF provides a "road map" that identifies the product according to its attributes, contents, compatibilities, dependencies and descriptions. See Chapter 10 for complete details on how to create a PSF.

Vendors or software packagers can define attributes that describe their software objects. These attributes, an essential part of the PSF, include such information as the product's short name (tag), a one-line full name (title) or a multiple paragraph description of the object. Other attributes include a multi-paragraph README file, a copyright information statement and others as described in Chapter 10.

You can also specify in the PSF what computer(s) and operating system(s) the product supports by using the `uname(1)` attributes of these system(s). These `uname` attributes are:

- The machine hardware model name (machine_type).
- The trade marked name of the operating system (os_name).
- The current release number of the operating system (os_release).
- The current version number of the operating system (os_version).

## Determining Product Structure

The next step in packaging software is to determine the product structure that your software should follow. SD-UX provides four levels of software objects:

| | |
|---|---|
| Filesets | Filesets include the actual product files, information that describes those files (attributes) and separate control scripts that are run before, during or after the product is installed, copied or removed. Filesets are the smallest manageable (selectable) software unit. (Filesets can only be members of a single *product* but can be part of many *bundles*.) |
| Subproducts | Subproducts are used to group related filesets within a product if the product contains several filesets. Subproduct definitions are optional. |
| Products | Software filesets (and/or subproducts) are usually grouped into collections that match the products that a customer purchases. The SD-UX commands maintain a product focus, while still allowing the flexibility to manage subsets of the products via subproducts and filesets. |

| | |
|---|---|
| Bundles | Bundles are collections of filesets that have been grouped together by HP for a specific purpose. For example, a series of filesets may be bundled for use on a specific, minimum-size workstation. Bundles are only provided by the HP factory. Vendors and system administrators cannot package in bundles at this time. |

**Note** Different versions of products can be defined for different platforms and operating systems, as well as different revisions (releases) of the product itself. Different versions can be included on the same distribution media.

## Phases of the Packaging Operation

A `swpackage` session consists of four phases:

- **Package Selection Phase**

  The system reads the **Product Specification File (PSF)** to determine the product, subproduct and filesets required for the structure; which files are contained in each fileset; and the attributes associated with these objects. The PSF is also checked for syntax errors at this time. If an error is encountered, the `swpackage` session terminates.
- **Analysis Phase**

  The system analyzes the packaging tasks and requirements BEFORE actually packaging the software to the recipient depot (tape). `swpackage` compares the software to be packaged against the recipient depot (tape) to make sure that the packaging operation can be attempted. If an error is encountered during analysis, the session terminates.
- **Packaging Phase**

  `swpackage` packages the source files and information into a product object, and creates/inserts the product into the distribution depot. If the depot does not exist, `swpackage` creates it (if you have permission to create depots) but it does not automatically register it like `swcopy` does. You must register the depot.

  If the target is a **tape media**, a temporary depot is created before the package is transferred to tape. You must also have permission to create this temporary depot.

- **Make Tape Phase**

  When packaging to a tape, the **make tape phase** actually copies the temporary depot to the tape.

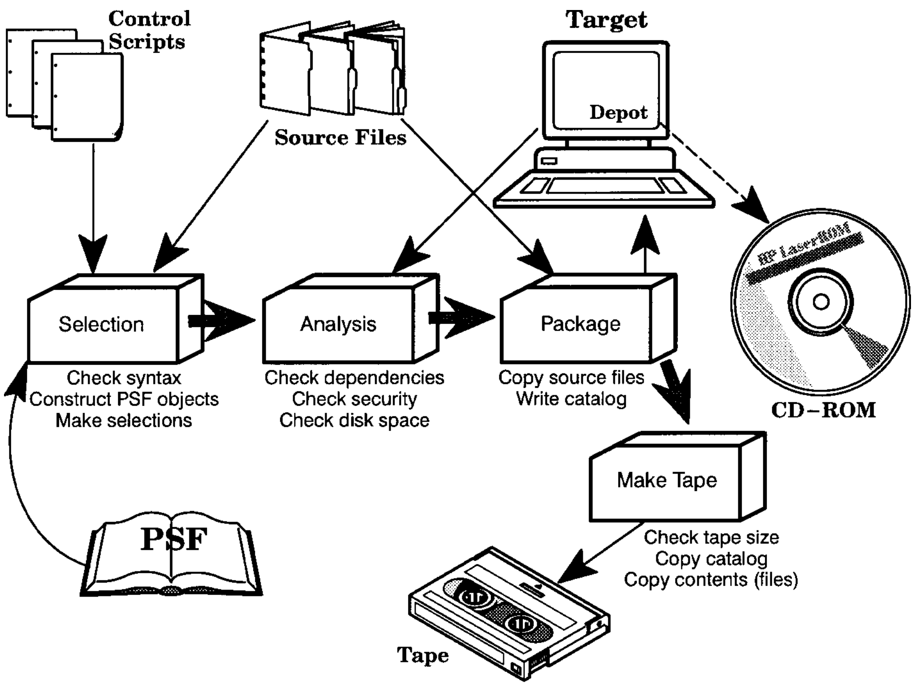


**Figure 9-1. An Overview of the Packaging Process**

## The Packaging Command

The `swpackage` command packages software products into a depot that can be accessed directly or used to create a CD-ROM/tape. You can also use this depot as a source to install or update from, using the `swinstall` command. The `swpackage` command provides only a command line user interface. There is no Graphical User Interface for the packaging tasks.

### Differences Between `swpackage` and `swcopy`

Both `swpackage` and `swcopy` commands create or change a depot. The differences between these commands are:

- The `swcopy` command merely copies filesets from an existing source (tape, CD-ROM or other depot) to a depot, while `swpackage` *creates products* (new collections of subproducts, filesets and control scripts) on a tape or depot.
- `swpackage` packages source files only on the local file system and builds them into a product for insertion only into a local depot or tape.
- After creating a depot, `swcopy` automatically registers that depot with the controller host so that it can be found by other commands. With `swpackage`, the depot is not registered until you explicitly invoke the `swreg` command afterward (see Chapter 4).
- `swpackage` can repackage products from a source depot to a tape. `swcopy` cannot copy to tape.

In summary, the `swpackage` command:

- organizes the software to be packaged into products, subproducts, and filesets;
- packages source files into filesets;
- sets permissions of the files being packaged;
- packages both a simple, one-fileset product as well as a complex product with many filesets (and many subproducts) and
- provides a way to repackage (change) existing products.

### Command Syntax

The syntax for `swpackage` is:

```
swpackage [-p] [-v[v]] [-V] [-s product_specification_file/directory]
          [-d directory|device] [-x option=value]
          [-X option_file] [-f software_file]
          [-C session file] [-S session file]
          [software selections] [@ target selections]
```

### A Simple Packaging Command

To package the software products defined in the PSF *product.psf* into the distribution depot */var/spool/sw* and preview the task at the "very verbose" level before actually performing it, type:

```
swpackage -p -vv -s product.psf -d /var/spool/sw
```

Of course, you could omit the `-d` */var/spool/sw* since you are packaging to the default depot directory. You can also use the `@ target` syntax to achieve the same result.

## Command Options

The `swpackage` command supports the following options:

| Option | Action |
|---|---|
| `-p` | Previews the specified package session without actually creating or modifying the distribution depot (tape). |
| `-v` | Turns on verbose output to stdout and lists messages for each product, subproduct and fileset being packaged. Adding a second v to the command (-vv) turns on "very verbose" output that displays all verbose messages plus messages for each file being packaged. (The `swpackage` logfile is not affected by this option.) SD-UX is shipped with -v enabled by default. To reduce messages to the minimum, change the `swpackage.`*verbose*= default option to 0. |
| `-V` | Lists the SD data model revisions that `swpackage` supports. By default, `swpackage` always packages with the latest data model revision. |
| `-s` *product_specification_file directory* | Specifies the Product Specification File (PSF) that describes all the bundles, products, subproducts, filesets and files used to package a depot or tape from a set of source files. Alternatively, it can specify an existing directory as the source for the packaging session. |
| `-d` *directory*\|*device* | Defines the location of the depot to be a directory when you are create a distribution depot (directory). The default depot directory is */var/spool/sw*.<br><br>If you are creating a tape, this option names the device file on which to write the `tar(1)` archive. You may want to set the *target_type* option to "tape" (`-x target_type=tape`). Without this option, `swpackage` uses the device file `/dev/rmt/0m` as the default. |
| `-x` *option*=*value* | Allows setting or changing an option on the command line. |

| | |
|---|---|
| `-f` *software file* | Reads the list of *software_selections* from a file. |
| `-X` *option file* | Uses the options in a specific options file. |
| `-C` *session file* | Run the command and save the current option and operand values to a file for re-use in another session. |
| `-S` *session file* | Run the command based on values saved from a previous session. |

You can also specify that the `swpackage` output be "piped" to an external command using the syntax:

```
swpackage -d "| <command>" -x target_type=tape -s <source name>
```

The `|` symbol must be quoted because it is interpreted by `swpackage` and not the shell.

This example shows how you can package a local source and write the output to a tape on a remote system ("host").

```
swpackage -d "| remsh host dd of=/dev/rmt/0m obs=10k" -x
target_type=tape -s <source>
```

## Command Operands

The `swpackage` command supports the standard *software_selections* operand: `bundle[.product[.subproduct][.fileset][,version]`.

If this specification is used in the command, `swpackage` packages ONLY those software selections from the full set as specified in the source PSF. In other words, the command line overrides the PSF.

If this specification is NOT used, `swpackage` packages ALL the products listed in the PSF.

See Chapter 2 for more information on operands.

## Extended Options and Defaults File

In addition to the command-line options listed above, you can change several `swpackage` behaviors and policy options by editing default values found in the defaults file */var/adm/sw/defaults*. This file controls defaults for the entire system. You can also list personal defaults in the *$HOME/.sw/defaults* file. Values in these file are specified using the `swpackage` *option=value* syntax.

See Appendix A for more information on these defaults.

The following options are recognized by `swpackage`:

| | |
|---|---|
| *source_type=file* | Defines the type of source used to package; allowed values are *file* and *directory*. |
| *source_file=psf* | Defines the default *product_specification_file* to read as the source when the source type (above) is *file*. The `-s` option overrides this default. |
| *source_directory= /var/spool/sw* | Defines the default directory to read as the source (when the source type is *directory*). The `-s` option overrides this default. |
| *target_type=directory* | Defines which type of distribution to create. The recognized types are *directory* and *tape*. |
| *target_directory= /var/spool/sw* | Defines the default directory in which products will be packaged. The `-d` option overrides this default when *target_type* is "directory". |
| *target_tape= /dev/rmt/0m* | Defines the default tape device file on which products will be packaged. The `-d` options overrides this default when *target_type* is "tape.". |
| *verbose=1* | Normally set to 1, a value of 0 (or no value) sends no messages (except errors and warnings) to stdout. A value of 1 sends "verbose" messages (for each product, subproduct and fileset being packaged). A value of 2 sends "very verbose" messages (verbose messages plus a message for each file packaged). The `-v` option overrides this default. |
| *loglevel=x* | Normally set to 1, a value of 0 sends no messages to the logfile. The default value of 1 sends "verbose" messages (a message for each product, |

|  |  |
|---|---|
|  | subproduct and fileset being packaged). A value of 2 sends "very verbose" messaging (verbose messages plus a message for each file being packaged). |
| *logdetail=false[true]* | Allows more control over the amount of information being sent to the logfile. A value of *true* allows POSIX events plus verbose messages (when used with *loglevel=1*) or all generated messages (when used with *loglevel=2*). The default value of *false* restricts the information to POSIX events only (with *loglevel=1*) or to POSIX and file level events only (with *loglevel=2*). |
| *logfile= /var/adm/sw/swpackage.log* | Defines the file where all messages are logged. |
| *follow_symlinks=false* | If "false", `swpackage` DOES NOT follow symbolic links in the package source files, but includes the symbolic links in the packaged product(s). If "true," follows symbolic links in the package source files and includes the file(s) they reference in the packaged product(s). |
| *include_file_revisions= false* | Controls whether `swpackage` includes each source file's revision attribute in the product(s) being packaged. Because this operation is time consuming, the revision attributes are not included by default. A value of "true" for this keyword causes `swpackage` to execute `what(1)` and possibly `ident(1)` (in that order) to try to determine a file's revision. |
| *enforce_dsa=true* | A value of "true" for this default causes `swpackage` to terminate if the disk space required to package the software selections into the recipient depot exceeds the file system's minfree threshold. A value of "false" causes `swpackage` to write into minfree space, up to the absolute limit of the file system. |
| *package_in_place=false* | A value of "true" for this keyword causes `swpackage` to build the specified products such that the distribution directory will not contain the |

|  |  |
|---|---|
|  | files that make up a product. Instead, `swpackage` inserts references to the original source files used to build a product. This lets you package products in a development or test environment without consuming the full disk space of copying all the source files into the distribution directory. |
| *reinstall_files=true* | Controls whether `swpackage` should re-install files that have not changed (in a fileset that already exists in the distribution directory and is being repackaged). A value of "false" for this keyword causes `swpackage` to only re-install those files that have changed (that is, the source file is different from the file in the depot). |
| *reinstall_files_use_cksum=false* | If set to "false," this keyword controls how `swpackage` determines if files that are being re-installed (re-packaged) have changed. By default, `swpackage` compares only the file's size and modification time. A value of "true" for this default also causes `swpackage` to compare the file's checksum values. |
| *create_target_acls= true* | If you are creating a distribution depot, this default determines whether `swpackage` will create Access Control Lists (ACLs) in the depot. If you are superuser and this option is set to "false," ACLs for each new product being packaged (and for the depot, if it is new) will NOT be created. When `swpackage` is invoked by any other user, it will always create ACLs in the distribution depot. This default has no impact on the ACLs that already exist in the depot. The `swpackage` command never creates ACLs when software is packaged onto a tape. |
| *write_remote_files= false* | This option controls whether `swpackage` writes to a recipient depot that exists on an NFS file system. If this option is set to "true" and superuser has "write" permission on the file system, then `swpackage` creates or changes a depot on that file system. |
| *media_capacity=1330* |  |

If you are creating a distribution tape, this default specifies the capacity of the tape in one million byte units (not Mbytes). This option is required if the media is not a DDS tape or a disk file. Without this option, `swpackage` sets the size to 1330 million bytes for tape and “free space up to minfree” on a disk file.

*share_link*= A *share_link* is an attribute which defines a directory to be sharable. A share link is an entry in the IPD (Installed Product Database) that defines a sharing point as exportable to another system (client). It is used by SD-UX for linkinstalls. Such directories are to be treated as read-only.

10

# Product Specification Files

The Product Specification File (PSF) drives the `swpackage` session. It describes how the product is structured and defines the attributes that apply to it.

Each SD-UX software object (bundle, product, subproduct, fileset and file) has its own set of **attributes** and each attribute has a **keyword** that defines it. A keyword is an identifier for the attribute, for example: `tag`, `title`, `revision`, `architecture`, etc. Most attributes are optional; they do not all need to be specified in the PSF. Each attribute has its own specific requirements, but the following rules apply:

- Keyword syntax is: `keyword` *value*.
- All keywords require one or more values, except as noted. If the keyword is in the PSF but the value is missing, a warning message is generated and the keyword is ignored.
- Comments can be placed on a line by themselves or after the keyword-value syntax. Comment lines are designated by preceding them with `#`.
- Use quotes when defining a value (for example, `description`) that can span multiple lines. Quotes are not required when defining a single-line value that contains embedded whitespace.
- Any errors encountered while reading the PSF causes `swpackage` to terminate. Errors are also logged to both `stderr` and the logfile.

## A Sample PSF

Here is a portion of a typical PSF file; it describes the product called "HP OpenView Software Distributor" or "SD":

```
distribution
   layout_version  1.0
# Vendor definition:
vendor
  tag           HP
  title         Hewlett-Packard Co.
  description   <data/description.hp
end
# Product definition:
product
  tag            SD
  title          HP OpenView Software Distributor
  revision       2.0
  number         J2326AA
  category       system_management
  category_title Systems Management Software
  description    <data/description
  copyright      <data/copyright
  readme         <data/README
  architecture   S700/S800_HPUX_9.0
  vendor_tag     HP
  machine_type   9000/[78]*
  os_name        HP-UX
  os_release     ?.09.*
  os_version     ?
  directory      /
  is_locatable   false
# Create a product script which executes during the swremove
# analysis phase.  (This particular script returns an ERROR,
# that prevents the removal of the product.)
  checkremove     scripts/checkremove.sd
 # Subproduct definitions:
  subproduct
    tag          Manager
    title        Management Utilities
    contents     commands agent data man
  end
  subproduct
    tag          Agent
    title        Agent component
    contents     agent data man
  end
 # Fileset definitions:
  fileset
```

```
        tag          commands
        title        Commands (management utilities)
        revision     2.15
        ancestor     OLDSD.MAN
        description  <data/commands
     # Control scripts:
        configure    scripts/configure.commands
     # Dependencies
        corequisites SD.data
     # Files:
        directory    ./commands=/usr/sbin
        file         swinstall
        file         swcopy
        .
        .
        .
        directory    ./nls=/usr/lib/nls/C
        file         swinstall.cat
        file         swpackage.cat
        file         swutil.cat
        directory    ./ui=/var/adm/sw/ui
        file         *
        .
        .
        .
     end # commands
       # other filesets
         .
         .
         .
     fileset
        tag          man
        title        Manual pages for the Software Distributor
        revision     2.05
        directory    ./man/man8=/usr/man/man8
        file         *
        directory    ./man/man4=/usr/man/man4
        file         *
        directory    ./man/man5=/usr/man/man5
        file         *
      end  #end of man fileset
    end  #end of product
```

From the above example, SD-UX can get all the information needed to package the product properly into a distribution depot.

In summary, the PSF can:

- define vendor information (optional) for bundles (including all products), or for individual products;
- specify one or more products;

- for each product, define attributes for one or more subproducts (optional), filesets and files; and
- define attributes for the distribution depot/media (optional).

## PSF Syntax

PSF syntax conforms to the `layout_version=1.0` of the POSIX 1387.2 Software Administration standard. Previous versions of SD-UX supported the POSIX `layout_version=0.8` syntax (which continues to be supported.) In the earlier version, the vendor is handled differently (see “Vendor Class” below) and the *corequisites* and *prerequisites* have singular titles (that is, corequisite and prerequisite). The `layout_version` option for `swpackage, swmodify, swcopy` and `swlist` controls which command SD writes (see Appendix A).

The values for each attribute keyword in your PSF must be of a specific type:

tag_string

Maximum length: 16 bytes

- A subset of ASCII characters
- Requires one or more alphanumeric characters (A-Z, a-z, 0-9), including the first character.
- White space characters are not allowed.
- Directory path character (/) is not allowed
- SD metacharacters (. , : #) are not allowed
- Shell metacharacters (;&(){}|<>“ ”‘ ’\) are not allowed

Examples: `HP, SD`

one_line_string

Maximum length: 80 bytes

- All ASCII characters.
- No white space characters, except for space and tab, are allowed.

Example: `Hewlett-Packard Company`

multi_line_string

Maximum length: Descriptions are 2048 bytes, Copyrights are 8192 bytes and READMEs can be 1 Mbyte maximum size.

- All ASCII characters in one or more paragraphs of text that can be specified

| | |
|---|---|
| | in-line, surrounded by double-quotes, or stored in a file and specified using a filename. |
| revision_string | Maximum length: 32 bytes<br>■ Contains dot-separated one_line_strings.<br>Examples: `2.0`, `A.09.00` |
| boolean | Maximum length: 5 bytes<br>■ Either "true" or "false." |
| path_string | Maximum length: 255 bytes for tapes, 1023 bytes for depots, 1023 bytes for source file pathnames.<br>■ An absolute or relative path to a file object.<br>Examples: `/usr`, `/mfg/sd/scripts/configure` |
| uname_string | Maximum length: 64 bytes for *machine_type*, 32 bytes for other `uname` attributes.<br>■ `uname` strings of ASCII characters only.<br>■ White space characters are not allowed.<br>■ Shell pattern matching notations ([]*?!) are allowed.<br>■ Values can be separated using the vertical bar (\|) to signify an "or" condition.<br>Examples: `9000/7*\|9000/8*`, `HP-UX`, `?.09.*`, `[A-Z]` |
| path_mapping_string | Maximum length: NA (Not Applicable)<br>■ A value of the form `source` [=*destination*] where the `source` is the directory where source files are located. The optional `destination` maps the source to a directory in which the files will actually be installed.<br>Example: `/build/hpux/700/mfg/usr=/usr` |
| file_specification | Maximum length: NA<br>■ Explicitly specifies a file to be packaged, using the format: |

```
[-m  mode] [-o [owner[,]][uid]] [-v] [-g [group[,]][gid]]
 [source] [destination]
```

- The source and destination can be paths relative to source and destination directories specified in the path_mapping_string.

  Example: `-m 04555 sbin/swinstall`
- You can use `*` to specify all files below the source directory specified in the *path_mapping_string*.

  Example: `file *`

permission_string

Maximum length: NA

- A value of the form:

```
[-m mode|-u umask] [-o[owner[,]][uid]] [-g[group[,]][gid]]
```

  where each component defines a default permissions value for every file defined in a fileset. The default values can be overridden in each file's specific definition. The owner and group fields are tag_strings. The uid and gid fields are unsigned integers. The mode and umask are unsigned integers that support only the octal character set: "0"-"7".

Example: `-u 0222 -o root -g sys`

software_specification

Maximum length: NA

- A software specification of the form:

```
bundle[.product][.subproduct][.fileset][,version]
```

  where the *version* has the form:

```
[r=revision][,a=arch][,v=vendor][,c=category][,instance_id]
```

Examples: `SD.agent` or
`SD,r=2.0,a=S700/800_HP-UX_9.0,v=HP`

version_component

Maximum length: NA

- These attributes are restricted to:
  - □ No whitespace characters are allowed.

    - Each version specification is repeatable within the component.
    - Specifications can be preceded by a logical operator (=, ==, >=, <=, <, >, or !=) which performs individual comparisons on dot-separated fields.
    - Pattern matching notations ([]*?!) are allowed only with the = logical operator.
- Attributes apply to `tag`, `revision`, `architecture`, `vendor_tag` and `category` keywords.

The table and sections on the next pages describe the syntax for the objects you can define in a PSF.

**Table 10-1. Keywords Used in the Product Specification File**

| Class | Keyword | Value | Example |
|---|---|---|---|
| Depot | `depot` | | |
| | `tag` | tag_string, 16 bytes | APPLICATIONS_CD |
| | `title` | one_line_string, 80 bytes | HP-UX Core Software Disk |
| | `description` | multi_line_string, 2048 bytes | </mfg/misc/depot/description |
| | `copyright` | multi_line_string, 8192 bytes | </mfg/misc/depot/copyright |
| | `number` | tag_string, 32 bytes | B2358-13601 |
| | `end` | | |
| Vendor | `vendor` | | |
| | `tag` | tag_string/version_comp., 16 bytes | HP |
| | `title` | one_line_string, 80 bytes | Hewlett-Packard Company |
| | `description` | multi_line_string, 2048 bytes | </mfg/misc/vendor/desc |
| | `end` | | |
| Product | `product` | | |
| (required) | `tag` (required) | tag_string/version_comp., 16 byte | SD |
| | `title` | one_line_string, 80 bytes | HP OpenView Software Dist. |
| | `revision` | rev._string/version_comp. 32 byte | 2.0 |
| | `description` | multi_line_string, 2048 bytes | </mfg/sd/data/description |
| | `vendor_tag` | tag_string, 16 bytes | HP |
| | `copyright` | multi_line_string, 8192 bytes | </mfg/sd/data/copyright |
| | `readme` | multi_line_string, 1 Mbyte | </mfg/sd/data/README |
| | `number` | tag_string, 32 bytes | |
| | `category` | tag_string/version_comp., 16 byte | J2326AA |
| | `category_title` | one_line_string, 80 bytes | systems management |
| | `is_locatable` | boolean, 5 byte | true |
| | `directory` | path_str./version_comp. 1023 bytes | /usr |
| | `architecture` | one_line_str./vers._comp. 80 byte | S700/800_HP-UX_9.0 |
| | `machine_type` | uname_string, 64 bytes | 9000/7*\|9000/8* |
| | `os_name` | uname_string, 32 bytes | HP-UX |
| | `os_release` | uname_string, 32 bytes | ?.09.* |
| | `os_version` | uname_string, 32 bytes | [A-Z] |
| | `end` | | |
| Subproduct | `subproduct` | | |
| | `tag` | tag_string, 16 bytes | Manager |
| | `title` | one_line_string, 80 bytes | SD Mgmt Interfaces Subset |
| | `description` | multi_line_string, 2048 bytes | </mfg/sd/data/manager/desc |
| | `contents` | one line list of fileset tag_strings | commands agent man data |
| | `end` | | |

**Table 10-1.**
**Keywords Used in the Product Specification File (continued)**

| Class | Keyword | Value | Example |
|---|---|---|---|
| Fileset (required) | `fileset` | | |
| | `tag` (required) | tag_string, 16 bytes | man |
| | `revision` | revision_string, 32 bytes | 1.42 |
| | `title` | one_line_string, 80 bytes | SD Manual Pages |
| | `description` | multi_line_string, 2048 bytes | </mfg/sd/data/man/desc |
| | `ancestor` | product.fileset | OLDSD.MAN |
| | `is_kernel` | boolean, 5 bytes | false |
| | `is_reboot` | boolean, 5 bytes | false |
| | `end` | | |
| File | directory | path_mapping_string | /build/hpux/700/mfg/usr=/usr |
| | file (required) | file_specification | -m 04555 sbin/swinstall or * |
| | file (required) | permission_string | -u 0222 -o root -g sys |
| | file_permiss. | | |
| Control Script | checkinstall | path_string, 1023 bytes | /mfg/sd/scripts/checkinstall |
| | preinstall | path_string, 1023 bytes | /mfg/sd/scripts/preinstall |
| | postinstall | path_string, 1023 bytes | /mfg/sd/scripts/postinstall |
| | configure | path_string, 1023 bytes | /mfg/sd/scripts/configure |
| | unconfigure | path_string, 1023 bytes | /mfg/sd/scripts/unconfigure |
| | verify | path_string, 1023 bytes | /mfg/sd/scripts/verify |
| | checkremove | path_string, 1023 bytes | /mfg/sd/scripts/checkremove |
| | preremove | path_string, 1023 bytes | /mfg/sd/scripts/preremove |
| | postremove | path_string, 1023 bytes | /mfg/sd/scripts/postremove |
| | control_file | path_string, 1023 bytes | /mfg/sd/scripts/subscripts |
| Dependency | prerequisites | software_specification | SD.agent |
| | corequisites | software_specification | SD.man |

## Depot Class

The Depot (tape) specification for a PSF looks like this:

```
depot
        tag                 APPLICATIONS_CD
        title               HP-UX Applications Software Disk
        description         </mfg/misc/depot/description
        copyright           </mfg/misc/depot/copyright
        number              B2358-13601
        [<vendor specification>]      optional
        <product specification>       required
        [<product specification>]     optional
end
```

Each keyword above defines an attribute of a distribution depot. The *depot* keyword is always required; all other attributes are optional.

| | |
|---|---|
| **tag** | The short name of the target depot (tape) being created/modified by `swpackage`. |
| **title** | The full name of the target depot (tape) being created/modified by `swpackage`. |
| **description** | The description of the target depot; either the text itself or a pointer to a filename that contains the text. |
| **copyright** | The text (or a pointer to a filename) for the copyright information for the depot's contents. |
| **number** | The part or manufacturing number of the distribution media (CD or tape depot). |
| **end** | Ends the depot specification, no value is required. This keyword is optional. If you use it and it is incorrectly placed, the specification will fail. |

The above attributes are designed for distribution tapes and CD-ROMs to allow you to list information about the media (that is, find out what media it is).

## Vendor Class

**Note** The `layout_version` keyword can affect how vendors are associated with products and bundles. If you specify a *layout_version* of *1.0*, the *vendor object* definition will only be associated with subsequent products and bundles that have matching `vendor_tag` attributes within those products or bundles, until another *vendor object* is defined. If another *vendor object* is defined, then subsequent `vendor_tags` must match that object's tag.

If you do not specify a `layout_version`, products and bundles will be automatically assigned a `vendor_tag` from the last *vendor class* you defined at the distribution level, or from a *vendor* that has been defined within a product or bundle (unless a `vendor_tag` with or without a value is defined). However, if a `vendor_tag` is defined (with or without a value), it takes precedence.

The **vendor** specification looks like this:

```
vendor
        tag          HP
        title      Hewlett-Packard Co.
        description  </mfg/misc/vendor/description
end
```

Each keyword defines an attribute of the vendor object. If a vendor specification is included in the PSF, `swpackage` requires only the `vendor` and `tag` keywords.

| | |
|---|---|
| **tag** | The vendor's identifier. |
| **title** | A one-line string that further identifies the vendor. |
| **description** | A multi-line description of the vendor. The description value can be either the text itself or pointer to a filename that contains the text. |
| **end** | An optional keyword that ends the vendor specification, no value is required. If you use it and it is incorrectly placed, the specification will fail. |

## Product Class

The Product specification looks like this:

```
  product
        tag                 SD
        revision            2.0
        title               HP OpenView Software Distributor
        description         </mfg/sd/data/description
        vendor_tag          HP
        copyright           </mfg/sd/data/copyright
        readme              </mfg/sd/data/README
        number              J2326AA
        category            system_management
        is_locatable        false
        directory           /usr
        architecture        Series_700/800_HP-UX_9.0
        machine_type        9000/7*|9000/8*
        os_name             HP-UX
        os_release          ?.09.*
        os_version          [A-Z]

        [<vendor specification>]          optional
        [<subproduct specification>]      optional
```

```
          :
          [<control_script specification>]   optional
          :
          <fileset specification>            required
          [<fileset specification>]          optional
          :
  end
```

For each product object specified, `swpackage` requires only the `product` and `tag` keywords, plus one or more fileset specifications.

| | |
|---|---|
| **tag** | The product's identifier (name). |
| **revision** | The revision information (release number, version) for the product. |
| **title** | A one-line string that further identifies the product. |
| **description** | A multi-line description of the product; either the text value itself or a filename that contains the text. |
| **vendor_tag** | Associates this product or bundle with the last defined vendor object, if that object has a matching *tag* attribute. |
| **copyright** | A multi-line description of the product's copyright; either the text value itself or a filename that contains the text. |
| **readme** | A text file of the README information for the product. |
| **number** | The part number of the product. |
| **category** | A 16-character tag that describes the product's category, such as "database," "spreadsheet" or "cad_package." |
| **category_title** | An 80-character phrase that further describes the product's category, such as "Systems Management Software," "Finite Element Analysis Package", etc. |
| **is_locatable** | Defines whether a product can be installed to any product directory, or whether it must be installed into a specific directory. The attribute can be set to "true" or "false." (If not defined, `swpackage` sets the default attribute to "false.") |
| **directory** | The default, absolute pathname to the directory in which the product will be installed. Defines the directory in which the product files are contained. |
| **architecture** | The target system(s) on which the product will run. Provides a human-readable summary of the four `uname` |

attributes (machine_type, os_name, os_release and os_version).

| | |
|---|---|
| **machine_type** | The machine type(s) on which the product will run. (If not specified, the keyword is assigned a wildcard value of "*", meaning it will run on all machines.) If there are multiple machine platforms, you must separate each machine designation with a \| (vertical bar). For example, a keyword value of `9000/7*\|9000/8*` means the product will run on all HP Series 9000 Model 7XX or all HP 9000 Series 8XX machines. Alternatively, the value `9000[78]*` would also work. |
| **os_name** | The operating system name on which the product will run. If not specified, the attribute is assigned a value of "*", meaning it will run on all operating systems. |
| **os_release** | The release number of the product's operating system. If not specified, the attribute is assigned a value of "*", meaning it will run on all releases. |
| **os_version** | The version number of the operating system. If not specified, the attribute is assigned a value of "*", meaning it runs on any version. |
| **end** | Ends the product specification, no value is required. This keyword is optional. If you use it and it is incorrectly placed, the specification will fail. |

## Subproduct Class

A Subproduct specification looks like this:

```
subproduct
       tag         Manager
       title       SD Management Interfaces Subset
       description </mfg/sd/data/manager/description
       contents    manager agent packager man doc
end
```

If a subproduct object is specified, `swpackage` requires the `subproduct`, `tag` and `contents` keywords.

| | |
|---|---|
| **tag** | The subproduct's identifier. |
| **title** | A one-line string that further identifies the subproduct. |

|  |  |
|---|---|
| **description** | A multi-line description of the subproduct; either the text value itself, or a filename that contains the text. |
| **contents** | A whitespace-separated list of the filesets that comprise the subproduct (that is, `contents` *fileset fileset fileset fileset...*).<br><br>In the PSF, fileset definitions are not contained within the scope of subproduct definitions. The **contents** keyword is used to assign filesets to subproducts. This linkage allows a fileset to be contained in multiple subproducts. |
| **end** | Ends the subproduct specification, no value is required. This keyword is optional. If you use it and it is incorrectly placed, the specification will fail. |

## Fileset Class

A Fileset Specification looks like this:

```
fileset
        tag           manB
        revision      1.42
        title         SD Manual Page
        description   </mfg/sd/data/man/description
        ancestor      OLDSD.MAN
        is_kernel     false
        is_reboot     false

        [<control script specification>]       optional
        :
        [<dependency specification>]           optional
        :
        <file specification>                   required
        [<file specification>]                 optional
        :
end
```

For each fileset object specified, `swpackage` requires the `fileset` and `tag` keywords, plus one or more file specifications.

|  |  |
|---|---|
| **tag** | The fileset identifier. |
| **revision** | The revision number or version of the fileset. |
| **title** | A one-line string that further identifies the fileset. |

|  |  |
|---|---|
| **description** | A multi-line description of the fileset; either the text value itself or a filename that contains the text. |
| **ancestor** | If the *swinstall.match_target* option is set to "true," this attribute causes the current fileset to be selected if the product.fileset specified as an `ancestor` exists on the target system. Note that the only items allowed in an ancestor attribute are the product and fileset names, separated by a period. Multiple ancestor attribute lines may be specified. |
| **is_kernel** | Defines a fileset to be a contributor to an operating system's kernel. The attribute can be set to "true" or "false." If not specified, `swpackage` sets this attribute to "false." |
| **is_reboot** | Declares that a fileset requires a system reboot after installation. The attribute can be "true" or "false." If not specified, `swpackage` sets this attribute to "false." If a local "swinstall" is done that entails a reboot, the system reboots both the agent, and more importantly, the controller so there is no process left to report SUCCESS or FAILURE. SD-UX does not do a "reconnect" after reboot. |
| **end** | Ends the fileset specification, no value is required. This keyword is optional. If you use it and it is incorrectly placed, the specification will fail. |

## Dependency Class

Dependencies can be specified between filesets within a product. Dependencies can also be specified between a fileset and another product, namely a subset of that product, a particular fileset within that product or the entire product. SD-UX supports two types of dependencies:

|  |  |
|---|---|
| *Corequisites* | A corequisite dependency states that a fileset requires the other software to operate correctly, but DOES NOT IMPLY ANY LOAD ORDER. (Run-time dependency). |
| *Prerequisites* | A prerequisite dependency states that a fileset requires the other software to be installed and/or configured correctly BEFORE it can be installed and configured. Prerequisites control the order of an installation with `swinstall` so that for each step in the install process, the prerequisite's actions are performed first. (This is called "install-time dependency".) |

The `swinstall`, `swcopy`, `swverify` and `swremove` commands understand dependencies. You can control whether dependencies must be present to install the software. By default, `swinstall` will not install unless dependencies can be met.

---

**Note** A dependency must always be specified using the full product identifier ([.subproduct][.fileset]) for the requisite software. You can also specify the revision number of the dependency using the exact revision number, revision ranges, wildcard characters, or comparision operators (=, ==, <, >, >=, <=, or !=, etc.). For example:

```
OS-CORE.CORE-SHLIBS,r>=B.10.00,r<=B11.0,r=Q.310
OS-CORE.CORE-SHLIBS,r=[AB].10.*
```

(Note that the asterisk wildcard character only works when the = operator is present.)

---

## Control Script Class

HP-UX supports execution of product and fileset **control scripts** that allow you to perform additional checks and operations beyond those supported. The `swinstall`, `swconfig`, `swverify` and `swremove` commands each execute one or more vendor supplied scripts. All these scripts are optional.

For a complete description of Control Scripts and guidelines for their use, see Appendix B.

## File Class

Within a fileset specification, you can specify the following file types to be packaged into the fileset by `swpackage`:

- control script
- regular file
- directory
- hard link
- symbolic link

Control scripts are fully described in Appendix B. If a recognized unpackageable file type or an unrecognized file type is given, an error message is printed.

If the file to be packaged is a link, then the link is created. See "Using `swinstall` to Update" in Chapter 2 for more information on how SD-UX handles files, directories and symbolic links.

The `swpackage` command supports these mechanisms for specifying the files contained in a fileset:

| | |
|---|---|
| directory mapping | You can point `swpackage` at a source directory in which the fileset's files are located. In addition, you can map this source directory to the appropriate (destination) directory in which this subset of the product's files will be located. |
| recursive (implicit) file specification | If directory mapping is active, you can simply tell `swpackage` to include recursively all files in the directory into the fileset. |
| explicit file specification | For all or some of the files in the fileset, you can name each source file and destination location. |
| default permission specification | For all or some of the files in the fileset, you can define a default set of permissions. |

These mechanisms can all be used in combination with the others.

## Recursive File Specification

The **file** * keyword directs `swpackage` to include every file (and directory) within the current source directory in the fileset. `swpackage` attempts to include the entire, recursive contents of the source directory in the fileset. (Partial wildcarding is not supported, e.g. `file dm*` to indicate all files starting with "dm".) See "Re-Specifying Files" for more information on changing attributes of files within filesets.

The `directory` keyword must have been previously specified before the `file *` specification can be used. If not, `swpackage` generates an error.

When processing the directory recursively, three kinds of errors may be encountered.

- Cannot search directory (permission denied)

- Cannot read the file (permission denied)
- Unsupported file type encountered

All attributes for the destination file object are taken from the source file, unless a `file_permission` keyword is active (this keyword is described below).

The user can specify multiple pairs of

```
directory   source[=destination]
file *
```

to gather all files from different source directories into a single fileset.

## Explicit File Specification

Instead of (or in addition to) the recursive file specification, you can explicitly specify the files to be packaged into a fileset (if the recursive inclusion is not desired, you do not specify the **file** * keyword.)

You can use the *directory* keyword to define a source (and destination) for explicitly specified files. If no directory keyword is active, then the full source path and the absolute destination path must be specified for each file.

An explicit file specification uses this form:

```
file [-m mode] [-o [owner[,]][uid]] [-g [group[,]][gid]] [-v] [source] [destination]
```

| | |
|---|---|
| `file` | This keyword specifies an existing file (usually within the currently active source directory) to include in the fileset. |
| `-m` *mode* | This option defines the (octal) mode for a file or directory. |
| `-o` *[owner[,]][uid]* | This option defines the destination file's owner name and/or uid. If only the *owner* is specified, then the owner and uid attributes are set for the destination file object, based on the packaging host's `/etc/passwd`. If only the *uid* is specified, it is set as the uid attribute for the destination object and no owner name is assigned. If both are specified, each sets the corresponding attribute for the file object.<br><br>During an installation, the owner attribute is used to set the owner name and uid, unless the owner name is not specified or is not defined in the target system's `/etc/passwd`. In this case, the uid attribute is used to set the uid. |
| `-g` *[group[,]][gid]* | This option defines the destination file's group name and/or gid. If only the *group* is specified, then the group and gid |

attributes are set for the destination file object, based on the packaging host's `/etc/group`. If only the gid specified, it is set as the gid attribute for the destination object and no group name is assigned. If both are specified, each sets the corresponding attribute for the file object.

During an installation, the group attribute is used to set the group name and gid, unless the group name is not specified or is not defined in the target system's `/etc/group`. In this case, the gid attribute is used to set the gid.

| | |
|---|---|
| `-v` | This option marks the file as "volatile," meaning it can be modified (i.e. deleted) after it is installed without impacting the fileset.<br><br>Files that may have their attributes (size, last modified time, etc.) changed through normal use after they are installed should be specified in the PSF file as "volatile" (by specifying `-v` on the line defining the file). `swverify` will not, by default, check file attributes for files that have the *is_volatile* attribute set to "true" (see the *check_volatile* option for `swverify`). |
| *source* | When specifying an existing source file, this value defines the path to it.<br><br>If this is a relative path, `swpackage` will search for it relative to the source directory set by the `directory` keyword. If no source directory is active, `swpackage` will search for it relative to the current working directory in which the command was invoked.<br><br>All attributes for the destination file object are taken from the source file, unless a `file_permission` keyword is active, or the `-m`, `-o`, or `-g` options are also included in the file specification. |
| *destination* | This value defines the destination path at which the file will be installed. If *destination* is a relative path, the active destination directory set by the directory keyword will be prefixed to it. If it is a relative path, and no destination directory is active, `swpackage` generates an error. If the destination is not specified, then the source is used as the destination, with the appropriate mapping done with the active destination directory (if any). |

When processing existing files in a source directory, four kinds of errors may be encountered.

- Cannot search directory (permission denied)
- Cannot read the file (permission denied)
- Unsupported file type encountered (other than a control script, regular file, directory, hard link or symbolic link)
- File does not exist

### Directory Mapping

The `directory` *source [= destination]* specification defines a source directory under which subsequently listed files are located. In addition, you can map the *source* directory to a *destination* directory under which the packaged files will be installed.

For example, the definition:

```
directory /build/hpux/700/mfg/usr = /usr
```

causes all files from the `/build/hpux/700/mfg/` directory to have the prefix `/usr` when installed. The destination directory must be a superset of the product's *directory* attribute, if defined. This attribute is defined by the `directory` keyword in the product specification. If the product' *directory* is defined, and the destination is not a superset, `swpackage` generates an error.

The *destination* directory must be an absolute pathname. If not, then `swpackage` generates an error.

The *source* directory can be either an absolute pathname, or a relative pathname. If relative, `swpackage` interprets it relative to the current working directory in which the command was invoked.

If the *source* directory does not exist, `swpackage` generates an error.

The `directory` specification is optional.

### Example File Specifications

The following examples illustrate the use of the `directory` and `file` keywords.

Include all files under `/build/hpux/700/mfg` to be rooted under `/usr`:

```
directory  /build/hpux/700/mfg=/usr
file  *
```

Include only certain files under `/build/hpux/700/mfg/`, to be rooted under `/usr` and `/var/adm/sw`:

```
directory /build/hpux/700/mfg=/usr

file sbin/swinstall
file sbin/swcopy
:
directory /build/hpux/700/mfg=/var/adm/sw

file nls/swinstall.cat  nls/en_US.88591/swinstall.cat
file defaults newconfig/defaults
file defaults defaults
```

Explicitly list files, no directory mapping specified:

```
file  /build/hpux/700/mfg/usr/bin/swinstall /usr/sbin/swinstall
file  /build/hpux/700/mfg/usr/bin/swcopy /usr/sbin/swcopy
file  /build/hpux/700/mfg/data/nls/swinstall.cat /var/adm/sw/nls/en_US.88591/swinstall.cat
file  /build/hpux/700/mfg/data/defaults /var/adm/sw/newconfig/defaults
file  /build/hpux/700/mfg/data/defaults /var/adm/sw/defaults
```

Use all specification types to include files:

```
directory  /build/hpux/700/mfg/usr=/usr
file *

directory  /build/hpux/700/mfg/data=/var/adm/sw
file  defaults newconfig/defaults
file  /build/hpux/700/mfg/data/defaults=/var/adm/sw/defaults
```

## Permission Specifications

By default, a destination file object will inherit the mode, owner, and group of the source file. The *file_permissions* keyword can be specified to set a default permission mask, owner, and group for all the files being packaged into the fileset:

`file_permissions [-m` *mode*`|` `-u` *umask*`] [-o [`*owner*`[,]] [`*uid*`]] [-g [`*group*`[,]][`*gid*`]]`

| | |
|---|---|
| `file_permissions` | This keyword only applies to the fileset it is defined in. Multiple *file_permissions* can be specified, later definitions simply replace previous definitions. |
| `-m` *mode* | This option defines a default (octal) mode for all file objects. |
| `-u` *umask* | Instead of specifying an octal mode as the default, you can specify an octal `umask (1)` value that gets |

"subtracted" from an existing source file's mode to generate the mode of the destination file.

By specifying a `umask`, you can set a default mode for executable files, non-executable files, and directories. (A specific mode can be set for any file, as described above.)

| | |
|---|---|
| `-o` *[owner[,]][uid]* | This option defines the destination file's owner name and/or or uid. See the discussion of the `-o` option in the "Explicit File Specification" section above. |
| `-g` *[group[,]][gid]* | This option defines the destination file's group name and/or or gid. See the discussion of the `-g` option in the "Explicit File Specification" section above. |

## Example Permission Specifications

The following examples illustrate the use of the *file_permission* keyword.

Set a readonly 444 mode for all file objects (requires override for every executable file and directory):

```
file_permissions  -m 444
```

Set a read mode for non-executable files, and a read/execute mode for executable files and directories:

```
file_permissions  -u 222
```

Set the same mode defaults, plus an owner and group:

```
file_permissions  -u 222  -o bin  -g bin
```

Set the same mode defaults, plus a uid and gid:

```
file_permissions  -u 222  -o 2  -g 2
```

Set the owner write permission in addition to the above:

```
file_permissions  -u 022  -o 2  -g 2
```

If you define no *file_permissions*, `swpackage` uses the default value `file_permissions -u 000` for destination file objects based on existing source files. (Meaning the `mode`, `owner/uid`, `group/gid` are set based on the source file, unless specific overrides are specified for a destination file.)

## Re-Specifying Files

In addition to specifying files as a group (with `file *`) for general attributes, the PSF also allows you to "re-specify" files *within* a fileset's definition to modify individual attributes.

For example, suppose you wanted to specify all the files in a fileset which contained 100 files. All these files were to be recursively "discovered" and packaged into the fileset. Most of them would have the same owner, group, and mode (and other file attributes).

But, out of those 100 files, there might be five that are volatile (that is, you don't care if they get modified or deleted). So, instead of listing all 100 files individually, and using the `-v` option for the five, you could specify all 100 with `file *` and then modify the five individually in their own way:

```
        directory source = /product
file *

        file -v 1
        file -v 2
        file -v 3
        file -v 4
        file -v 5
```

This also works well for permissions. If nearly all the 100 files above had the same permission attributes, but three should have a different owner and mode, you could specify:

```
directory  source = /product

file_permissions -o bin -g bin -m 555
file *

file_permissions -o root -g other -m -04555
file 1
file 2
file 3
```

Essentially, this capability combines the recursive file specifications function with explicit file specification.

# 11

# Analyzing and Building Packages

There are four checks performed on your packaging process during the Package Analysis Phase:

1. **Check for unresolved dependencies.**

   For every fileset in each selected product, `swpackage` checks to see if a requisite of the fileset is NOT also selected or NOT already present in the target depot. Any unresolved dependency within the product will generate an ERROR. An unresolved dependency across products would produce a NOTE.

2. **Check your authorization to package (or re-package) products.**

   For each new product (that is, a product that does NOT exist on the target depot) `swpackage` checks the target depot to see if you have permission to create a new product on it (that is, "insert" permission). If you do not, the product is not selected.

   For each existing product (that is, one you are re-packaging) `swpackage` checks to see if you have permission to change it (that is, "write" permission). If you do not, the product is unselected.

   If all products are not selected because permission is denied, the session terminates with an ERROR.

   If the depot is a new depot or if you are packaging to a tape, this authorization check is skipped. If you have permission to create a new depot, then you have permission to create products within it. And since a tape session first writes to a temporary depot then copies it to tape, if you have permission to create a new (temporary) depot, you can package to tape.

3. **Check for software being repackaged.**

   For each selected product, `swpackage` checks to see if the product already exists in the target depot.

a. If it does exist, `swpackage` checks to see which filesets are being added (new filesets) or modified.
b. If it exists and all filesets are selected, `swpackage` checks to see if any existing filesets have been obsoleted by the new product.

4. **Disk Space Analysis (DSA)**

   `swpackage` will check to see if you have enough free disk space on the target depot to package the selected products.

   There are three possible results:

   a. Part of the package will encroach into minfree space on the disk, causing an ERROR.
   b. The package phase will require more space on the disk than is available, causing an ERROR.
   c. A NOTE will show the impact the packaging phase will have on available disk space.

   The disk space analysis check is not performed for tape targets. Instead, the Copy to Tape Phase does a "tape space calculation" to ensure that the tape can accommodate all the software being packaged. If one tape cannot hold it all, then `swpackage` will partition the software across multiple tapes.

   See the section "Advanced Topics for `swpackage`" for information on how to override disk space analysis.

## The Package Building Phase

When packaging a product, if the target depot does not exist, `swpackage` creates it. If it does exist, `swpackage` will merge new products into it. For each different version of the product, a directory is created using the defined product tag attribute and a unique instance number (instance_ID) for all the **product versions** that have the same tag.

Before a new storage directory is created, `swpackage` checks to see if this product version has the same identifying attributes as an existing product version.

If all the identifying attributes match, you are re-packaging (modifying) an existing version. Otherwise, `swpackage` creates a new version in the target distribution.

The packaging process uses an explicit ordering to avoid corrupting the target distribution if a fatal error occurs. Each product is packaged in its entirety and when all specified products have been packaged successfully, the distribution's global **INDEX** file is built/rebuilt. Within each product construction, the following order is adhered to:

1. Check to see if the product is new or already exists. If it is new, create the product's storage directory.
2. For each fileset in the product, copy the fileset's files into their storage location (within the product's storage directory), and create the fileset's catalog (database information) files.
3. After the individual filesets, create the product's informational files (meta-files).

A target depot is only the first step in creating a CD-ROM. If the ISO 9660 standard format is desired, a utility to perform this conversion would be necessary. This conversion is not supported by `swpackage`.

Distribution tapes are created in `tar (1)` format. To create the tape, `swpackage` first builds the products catalog in a temporary distribution depot (it is removed when `swpackage` completes.) After the distribution depot is constructed, `swpackage` then archives the catalog, along with the real files, onto the tape device.

When archiving a product that contains **kernel filesets** onto a tape media, `swpackage` will put these filesets first within the archive to provide efficient access by `swinstall`. `swpackage` also orders filesets based on prerequisite dependency relationships.

## Modifying an Existing Software Product

The `swpackage` command allows you to add to or change an existing product in a depot by changing the Product Specification File that was used to package the existing product. You can then invoke `swpackage` and specify the appropriate *software_selections* on the command-line. Each specified software selection should correspond to a modified product, subproduct, or fileset definition within the PSF file. If new filesets are being added to an existing product, `swpackage` will identify the product and add the fileset(s). If product, subproduct or fileset attributes are being modified, `swpackage` will match them and make the replacements.

If a modified product, subproduct, or fileset specification redefines any attributes, the new attribute values will replace the existing values.

## Output of Logfile Messages

The log file `/var/adm/sw/swpackage.log` captures the output from the `swpackage` session. `swpackage` by default logs messages at the verbose level.

A sample log:

```
=======  05/11/93 17:09:28 MDT  BEGIN swpackage SESSION

      * Session started for user "rmr@moab.fc.hp.com".

      * Source:        moab:sd.psf
      * Target:        moab:/dev/rmt/0m
      * Software selections:
            SD.agent
      * Options:
            preview                     false
            verbose                     1
            loglevel                    1
            logfile                     /var/adm/sw/swpackage.log

            source_type                 file
            target_type                 tape
            media_capacity              1330 (million bytes)

            package_in_place            false
            follow_symlinks             false
            include_file_revisions      false
            enforce_dsa                 true
            reinstall_files             true
            reinstall_files_use_cksum   false
            write_remote_files          false
            create_target_acls          true

      * Beginning Selection Phase.
      * Reading the Product Specification File (PSF) "sd.psf".
      * Reading the product "SD" at line 7.
      * Reading the fileset "manager" at line 20.
      * Reading the fileset "agent" at line 60.
      * Reading the fileset "packager" at line 90.
      * Reading the fileset "man" at line 120.
      * Reading the fileset "examples" at line 140.

NOTE:    The temporary target depot "/usr/tmp/pkgAAAa21590" has been created.
      * Selection Phase succeeded.

      * Beginning Analysis Phase.
      * Checking file path length restrictions.
      * Checking software dependencies.
```

```
        * Checking security restrictions.
        * Checking for existing products and filesets.
        * Analysis Phase succeeded.

        * Beginning Package Phase.
        * Packaging the product "SD".
        * Packaging the fileset "SD.agent".
        * Package Phase succeeded.

        * Beginning Tapemaker Phase.
        * Copying the temporary depot to the tape "/dev/rmt/0m".
        * Calculating the tape blocks required to copy the temporary depot to
          the tape "/dev/rmt/0m".
NOTE:     The temporary depot requires 2536 Kbytes on the tape "/dev/rmt/0m".

        * Writing the tape "/dev/rmt/0m" (tape 1 of 1).

        * Removing the temporary depot.
        * Tapemaker Phase succeeded.

=======  05/11/93 17:10:44 MDT  END swpackage SESSION
```

In summary, message logging for `swpackage`:

- sends "verbose" messages to `stdout`.

  You can set the *verbose=* option to 0 which causes reduced output to be sent to `stdout`.
- sends ERRORS/WARNINGS to `stderr`.
- no logfile messages are written in preview mode.
- the logfile is equal to `stdout` plus `stderr`.

## Advanced Topics for `swpackage`

The topics discussed in this section are for those who have a firm understanding of the SD-UX packaging process and who want to have more control over it.

## Packaging Security

SD-UX provides Access Control Lists (ACLs) to authorize who has permission to perform specific operations on depots. Because the `swpackage` command creates and modifies local depots only, the SD-UX security provisions for remote operations do not apply to `swpackage`. See Chapter 8 for more information on ACLs.

The `swpackage` command is *setuid root*, that is, the Package Selection phase operates as the invoking user, the Analysis and Packaging phases operate as the superuser. As superuser, you owns and manages all depots and therefore have permissions for all operations on a depot. (Remember that even if you are superuser, you still need the appropriate write permissions if the depot happens to be on an NFS volume on another system.)

If you are not the local superuser, you will not have permission to create or modify a depot unless the local superuser grants you permission.

`swpackage` checks and enforces the following permissions:

1. **Can you create a new depot?**

| | |
|---|---|
| Superuser | Yes |
| Non-superuser | Yes, if the ACL for the local host grants the user “insert” permission, i.e. permission to insert a new depot into the host.<br><br>If the proper permissions are not in place and the depot is a new one, `swpackage` terminates with an error. |

2. **Can you create a new product?**

| | |
|---|---|
| Superuser | Yes |
| Non-superuser | Yes, if the depot is new and you passed check #1 above or if the ACL for an existing depot grants you “insert” permission, i.e. permission to change the contents of the depot (by adding a new product).<br><br>If you are denied authorization to create a new product, `swpackage` generates an error message and excludes the product from the session. |

3. **Can you modify an existing product?**

| | |
|---|---|
| Superuser | Yes |
| Non-superuser | Yes, if the ACL for the existing product grants you "write" permission, i.e. permission to overwrite/change the contents of the product. If you are denied authorization to change an existing product, `swpackage` generates an error message and excludes the product from the session.<br><br>If you are denied "insert" and "write" permission for all selected products, `swpackage` terminates with an error. |

4. **Can you change the depot-level attributes?**

| | |
|---|---|
| Superuser | Yes |
| Non-superuser | Yes, if the depot is a new one and you passed check #1 above or if the ACL for an existing depot grants you "write" permission, i.e. permission to write/change the contents of the depot (same as #2 above).<br><br>If you are denied authorization to change an existing depot and if the PSF specifies some depot-level attributes, then `swpackage` produces a warning message and does not change the depot attributes. |

## ACL Creation

When `swpackage` creates a new depot or a new product, it also creates an ACL for it:

| | |
|---|---|
| New depot | `swpackage` creates an ACL for the depot and a template ACL for all the products which will be packaged into it.<br><br>The depot ACL is generated from the the host's *global_soc_template* ACL (that is, the template ACL established for new depots and new root file systems).<br><br>The depot's *product_template* ACL is generated from the host's *global_product_template* ACL (that is, the host's template ACL for new products).<br><br>The user running `swpackage` is established as the owner of the new depot and is granted permissions as defined in the depot ACL (which come from the *global_soc_template*). |

| | |
|---|---|
| New product | `swpackage` creates an ACL for the product; the ACL is generated from the depot's *product_template* ACL.<br><br>ACL creation can be disabled (as described in another Advanced Topic below). When no ACL exists for a depot, only the superuser can create new products or add/modify depot attributes. When no ACL exists for a product, only the superuser can modify it. |

## Repackaging

**Note** For more information on repackaging, refer to the details on modifying an existing product found in an earlier section of this chapter.

There are two types of repackaging:

1. Adding to or modifying a fileset in an existing product.

   Editing the PSF by adding a new fileset definition or changing an existing fileset's definition.

   Running `swpackage` on the edited PSF, specifying the new/changed fileset on the command line:

   ```
   swpackage -s psf -d depot <other options> product.fileset
   ```

   This invocation works regardless of whether subproducts are defined in the product.

   If you change a fileset by changing it's tag attribute, `swpackage` cannot correlate the existing, obsolete fileset with the new fileset. Both become part of the changed product. To get rid of the obsolete (renamed) fileset, use `swremove`:

   ```
   swremove -d  product.old_fileset @ depot
   ```

2. Modifying an entire existing product.

   Editing the PSF by adding new fileset definitions, changing existing fileset definitions, deleting existing fileset definitions or changing the product's definition (product-level attributes).

   Running `swpackage` on the PSF, specifying the product on the command-line:

   ```
   swpackage -s psf  -d depot  <other options>  product
   ```

If you have deleted some fileset definitions in the PSF or modified a fileset by changing it's tag attribute, `swpackage` will produce warning messages about the existing filesets that are not part of the modified product's definition (in the PSF). The existing filesets plus the new filesets in the product's definition (in the PSF) will all be contained in the modified product.

The warnings are produced during analysis phase, and are only produced when the whole product is being repackaged (as opposed to subsets of the product).

To get rid of the obsolete (renamed) filesets, use `swremove`:

```
swremove -d  product.old_fileset @ depot
```

You may want to `swremove` the product entirely before repackaging the changes:

```
swremove -d  product @ depot
swpackage -s psf  -d depot  <other options>  product
```

## Packaging In Place

If you specify the option `-x` *package_in_place=true*, `swpackage` will package each of the specified products such that the source files are not copied into the target depot. Instead, `swpackage` inserts references to the source files that make up the contents of each fileset. Control scripts are always copied.

This feature lets you package products in a development or test environment without consuming the full disk space of copying all the source files into the target depot. Disk space analysis is skipped when the *package_in_place* option is "true."

The source files must remain in existence—if some are deleted, then any operations that use the depot as a source (for example, installing the product with `swinstall`) will fail when they try to access the missing source file(s).

If a source file changes and the product is not repackaged, the information that describes the source file will be incorrect (for example, the file *checksum*). This incorrect information will not prevent the use of that target depot as a source (for example, installing with `swinstall`). However, the incorrect information will be propagated along each time the product is copied or installed from the depot. The result will be that a `swverify` of the installed product will always flag the inconsistencies with an ERROR (unless you disable the check of file contents).

## Following Symbolic Links in the Source

If you specify the option `-x` *follow_symlinks=true*, `swpackage` will follow every source file that is a symbolic link and include the file it points to in the packaged fileset.

`swpackage` will also follow each source directory that is a symbolic link—which will affect the behavior of the "file *" keyword (recursive file specification). Instead of including just the symbolic link in the packaged fileset, the directory it points to and all files contained below it will be included in the packaged fileset.

The default value for this option is "false," which causes symbolic links that are encountered in the source to be packaged as symbolic links. The symbolic link can point to a file that is also part of the fileset, or to a file that is not.

## Generating File Revisions

If you specify the option `-x` *include_file_revisions=true*, `swpackage` will examine each source file using the `what(1)` and `ident(1)` commands to extract an SCCS or RCS revision value and assign it as the file's revision attribute.

Because a file can have multiple revision strings embedded within it, `swpackage` uses the first one returned. It extracts the revision value from the full revision string and stores it.

This option is time consuming, especially when a `what(1)` search fails and the `ident(1)` command is then executed.

The default value for this option is "false," which causes `swpackage` to skip the examination. No value for the revision attribute is assigned to the files being packaged.

## Overriding Disk Space Analysis Errors

The `swpackage` command performs a Disk Space Analysis (DSA) during the analysis phase. If the file system(s) that contain the target depot do not have enough free space to store the products being packaged, `swpackage` will print an error message and then terminate the session. It checks both the absolute free space threshold, and the minimum free threshold (minfree). Crossing either threshold will generate the error condition.

If you specify the option `-x` *enforce_dsa=false*, `swpackage` will change the error to a warning and continue. This flexibility lets you cross into the minfree space to complete a packaging operation.

## Writing to Multiple Tapes

When you package products to a distribution tape, the `-x` *media_capacity* option defines the size of the tape media (in one million byte units). The default value for this option is *media_capacity=1330*, which is the size of an HP DDS tape. If the target tape is not a DDS tape, you must specify the *media_capacity* value.

**Note** The capacity of the DDS tape is in one million byte units (1,000,000 bytes), NOT Mbyte units (1,048,576 bytes). Most tape drive manufacturers specify capacity in one-million byte units.

If the products being packaged require more space than the specified media capacity, `swpackage` will partition the products across multiple tapes.

To find out if multiple tapes will be required, `swpackage` will calculate the tape blocks required to store the depot catalog and each product's contents.

When multiple tapes are necessary, `swpackage` will write the entire catalog on to the first tape plus any product contents that will also fit. For each subsequent tape, `swpackage` will prompt you for a "tape is ready" response before continuing.

To continue with the next tape, enter one of the following responses:

| | |
|---|---|
| Return | Use the same device. |
| pathname | Use the new device/file "pathname". |
| *quit* | Terminate the write-to-tape operation. |

Partitioning is done at the fileset level, so a given product can span multiple tapes. A single fileset's contents cannot span multiple tapes. If any single fileset has a size that exceeds the media capacity, `swpackage` generates an error and terminates. It also generates an error if the catalog will not fit on the first tape.

## Making Tapes from an Existing Depot

You can copy one or more products from an existing depot to a tape using `swpackage`. Instead of specifying a Product Specification File as the source for a packaging session, just specify an existing depot. For example:

```
swpackage -s /var/spool/sw  ...
```

To copy all of the products in a depot to a tape:

```
swpackage -s depot -d tape -x target_type=tape
```

To copy only some of the products in a depot to a tape, specify the products as software selections:

```
swpackage -s depot -d tape -x target_type=tape product product ...
```

The `-f` *file* option can be used to specify several software selections (as listed in the *file*) instead of listing them on the command line.

When products are copied from a depot to a tape, the ACLs within the depot are NOT copied. (The `swpackage` command never creates ACL's when software is packaged onto a tape.)

## Registering Depots Created by `swpackage`

When a new depot is created by `swpackage`, it is NOT automatically registered with the local host's **swagentd** daemon. To register the depot, you must execute the `swreg` command:

```
swreg -l depot @  <depot name>
```

Registering a depot makes it generally available as a source for `swinstall` and `swcopy` tasks. See Chapter 4 for more information on registering depots.

In general, a depot that is not registered can be used as a source depot only by the superuser or the local user who created the depot. The depot is also not available for remote operations (for example, "pull" operations). This SD-UX restriction prevents someone from creating a "Trojan Horse" depot with a process other than `swpackage`. If this depot was then used as the source for `swinstall` operations to other systems, it could cause the "Trojan Horse" to be installed on every target system and potentially activated.

Registration provides a type of "public recognition" for the packaged depot:

- you can see the depot in the `swinstall/swcopy` GUI and see it in `swlist` depot-level listings.

- you can read products from the depot (for example, to install).

If the only uses of a depot created with `swpackage` are local accesses by the packaging user, depot registration is not required.

## Creating a Depot and Mastering it to a CD-ROM

When `swpackage` creates a new depot or packages a new product, it always creates an ACL for the depot/product. If you were to to create a depot and then master it onto a CD-ROM, the CD-ROM would contain all those ACLs which could cause the following problems:

- it may result in too-restrictive permissions on the CD-ROM depot.
- you could have too many "user-specific" ACLs on the CD-ROM.

To solve these problems, you can tell `swpackage` to NOT create ACLs in the depot by setting the *create_target_acls* default to "false."

This feature is provided only for the superuser because only the local superuser can change, delete, or add ACLs to a depot that has no ACLs. The local superuser always has all permissions.

The `-x` *create_target_acls=false* default causes `swpackage` to skip the creation of ACLs for each new product being packaged (and for the depot, if it is new). This option has no impact on the ACLs that already exist in the depot.

When a depot is used as a source for other SD-UX operations, its ACLs (or lack thereof) have no bearing on the ACLs created for the targets of the operation. Source ACLs are not related to target ACLs.

The `swpackage` command never creates ACLs when software is packaged onto a tape.

## Depots on Remote File systems

Because the `swpackage` analysis and build phases operate as the superuser, there are constraints on how `swpackage` creates, adds to, or modifies products on a depot that exists in an NFS-mounted file system.

If the superuser DOES NOT have write permission on the remote file system, then `swpackage` will be unable to create a new depot - it will terminate before the analysis phase begins.

If the superuser DOES have write permission on the remote file system but the option *write_remote_files* is "false," then `swpackage` will be unable to create a new depot - it will terminate before the analysis phase begins.

If the superuser DOES have write permission on the remote file system and you specify the option `-x` *write_remote_files=true*, then `swpackage` will create the new depot and package products into it.

The constraints for an existing NFS mounted depot are the same as when creating a new depot.

So, you must:

1. Set the *write_remote_files* option to "true" and
2. Make sure the superuser can write to the NFS file system to package a depot on an NFS-mounted file system.

When these constraints are satisfied, the SD-UX ACL protection mechanism will control operations on NFS mounted depots the same way it controls operations on local depots.

# Appendices

A

# Default Options and Keywords

SD-UX command behaviors and policy options may be changed by modifying default values found in */usr/lib/sw/sys.defaults* and storing them in the system level (`/var/adm/sw/defaults`) or user level (`$HOME/sw/defaults`) defaults files. Values in these defaults are specified using this syntax:

```
command.option=value
```

For example, to change the *use_alternate_source* default from "false" to "true" for an installation, edit the defaults file to include this line:

```
swinstall.use_alternate_source=true
```

These values can also be overridden by specifying an *options file* with the `-X` option (same format as the defaults files) or with the `-x` *option=value* option directly on the command line. They can also be changed using the GUI Options Editor.

Altering default values and storing them in a defaults file can help when you want the command to behave the same way each time the command is invoked.

## Defaults Listed Alphabetically

**agent=/usr/lbin/swagent**

This is the default location of the executable invoked to perform agent tasks.

**agent_auto_exit=true**

Causes the target agent to automatically exit after Execute phase, or after a failed Analysis phase. This is forced to `false` when the controller is using an interactive UI, or when -p (preview) is used.

Enhances network reliability and performance.

The default is `true` - the target agent will automatically exit when appropriate.

If set to `false`, the target agent will not exit until the controller explicitly ends the session.

**agent_timeout_minutes=10000**

Causes a target agent to exit if it has been inactive for the specified time.

You can use this default value to make target agents more quickly detect lost network connections. (RPC can take as long 130 minutes to detect a lost connection.) The recommended value is the longest period of inactivity expected in your environment.

For command line invocation, a value between 10 and 60 minutes is recommended when a graphical interface will be used.

The default is 10,000 minutes, which is slightly less than seven days.

**allow_downdate=false**

Normally set to "false," installing an older version of software than already exists is disallowed. This keeps you from installing older versions by mistake. Additionally, many software products will not support this "downdating."

If set to "true," a previous version can be installed but a warning message will still be issued.

Applies to `swinstall`.

**allow_incompatible=false**

Normally set to "false," only software compatible with the local host is allowed to be installed or configured.

If set to "true," no compatibility checks are made.

Applies only to `swinstall, swverify, swconfig`.

**allow_multiple_versions=false**

Normally set to "false," installed or configured multiple versions (for example, the same product, but a different revision, installed into a different location) are disallowed. Even though multiple *installed* versions of software are supported if the software is locatable, multiple *configured* versions will not work unless the product supports it.

If set to "true," you are allowed to install and manage multiple versions of the same software.

Applies to `swinstall, swconfig, swverify.`

**alternate_source =**

Syntax is `host:/path`, used when *use_alternate_source* default is set to "true."

By default, this option is not defined, meaning use the same path as the controller. See the *use_alternate_source* default.

**auto_kernel_build = true[false]**

Normally set to true. Specifies whether the removal of a kernel fileset should rebuild the kernel or not. If the kernel rebuild succeeds, the system will automatically reboot. If set to false, the system will continue to run the current kernel.

If the *auto_kernel_build* option is set to "true", the *autoreboot* option must also be set to "true". If the *auto_kernel_build* option is set to "false" the value of the *autoreboot* option does not matter.

Applies to `swremove` only.

**autoreboot = false**

Normally set to "false," installation of software requiring a reboot is not allowed from the command line.

If set to "true," this option allows selection of the software and automatically reboots the local host.

Applies to `swinstall, swremove`

**autorecover_product = false**

Normally "false," files are removed as they are updated by `swinstall.` If there is a load error, the product loading will be marked CORRUPT and the install must be re-tried.

When this option is "true," all files are saved as backup copies until all filesets in the current product loading are complete; then they are removed. At the cost of a temporary increase in disk space and slower performance, this allows for automatic recovery of filesets in that product if the load fails.

Note: This autorecover operation will not work properly if any software has pre-install scripts that move or remove files.

Applies only to `swinstall.`

**autoselect_dependencies = true**

Causes SD-UX to automatically select requisites when software is being selected. When set to true, and any software which has requisites is selected, it makes sure that the requisites are met. If they are not already met, they are automatically selected for you. If set to false, automatic selections are not made to resolve requisites. This option does not apply to `swconfig -u`.

Applies to `swinstall`,`swcopy`, `swverify`, `swconfig`.

**autoselect_dependents = false**

Causes `swconfig` to automatically select dependents when software is being selected. When set to true, and any software on which other software depends is selected, SD-UX makes sure that the dependents are also selected. If they are not already selected, they are automatically selected for you. If set to false, dependents are not automatically selected.

Applies to `swconfig -u` and `swremove`..

**autoselect_reference_bundles = true**

If "true," a bundle that is "referenced" will be installed along with the software from which it is made.

If "false," the software can be installed without the bundle that contains it.

Applies to `swinstall, swcopy` and `swremove`.

**check_contents = true**

Normally set to "true," verify *mtime*, *size* and *cksum* of files.

Applies only to `swverify`.

**check_permissions = true**

Normally set to "true," verify owner, uid, group, gid and mode attributes of files.

Applies only to `swverify`.

**check_requisites = true**

Normally set to "true," verify that the prerequisites and corequisites of filesets are being met.

Applies only to `swverify`.

**check_scripts = true**

Normally set to "true," run the vendor-supplied verify scripts when installing software.

Applies only to `swverify`.

**check_volatile = false**

Normally set to "false," include installed files in the verification that have the *is_volatile* attribute set. By default, installed volatile files do not have their attributes verified because they are volatile by definition.

Applies only to `swverify`.

**codeword = xxxxx**

This default allows you to enter your codeword for the software licensing procedure. Once entered you need not re-enter the codeword.

**compress_cmd = gzip**

This is the command called by the source agent to compress files before transmission. The default value is */usr/contrib/bin/gzip*.

**compress_files = false**

Normally set to "false," files will not be compressed before transfer from a remote source and uncompressed after transfer.

If set to "true," compressing files will enhance performance on slower networks (50 Kbytes/sec. or less) However, it may not improve fast networks.

Applies to `swinstall`, `swcopy` and `swpackage`.

**compression_type = gzip**

Defines the default compression type used by the agent (or set by `swpackage`) when it compresses files during or after transmission.

If *uncompress_files* is set to "false," the compression type is recorded for each file compressed so that the correct uncompression can later be applied during a `swinstall`, or a `swcopy` with *uncompress_files* set to "true."

The *compress_cmd* specified must produce files with the `compression_type` specified.

The *uncompress_cmd* must be able to process files of the `compression_type` specified unless the format is "gzip" which is uncompressed by the internal uncompressor ("funzip"). To use "gzip" you must load the SW-DIST.GZIP fileset. If the SW-DIST.GZIP fileset (which is optional freeware) is loaded, then you may set the compression options to:

```
compress_cmd=/usr/contrib/bin/gzip
uncompress_cmd=/usr/contrib/bin/gunzip
compression_type=gzip
```

Applies to `swpackage` and `swagent`.

**control_files =**

When adding or deleting control file objects, this option lists the tags of those control files. There is no supplied default. (Control file objects being added can also be specified in the given product_specification_file .)

If there is more than one tag, they must be separated by whitespace and surrounded by quotes.

Applies only to `swmodify`.

**create_target_acls = true**

Normally set to "true," this default determines whether `swpackage` will create Access Control Lists (ACLs) in the depot.

If you are superuser and this option is set to "false," ACLs for each new product being packaged (and for the depot, if it is new) will NOT be created.

When `swpackage` is invoked by any other user, it will always create ACLs in the distribution depot. This default has no impact on the ACLs that already exist in the depot. The `swpackage` command never creates ACLs when software is packaged onto a distribution tape.

**create_target_path = true**

Normally set to "true," creates the target directory if it does not already exist.

If "false," target directory is not created. This option can be used to avoid creating new depots by mistake.

Applies only to `swinstall, swcopy`.

**customer_id = xxxxxx**

This default allows you to enter your customer identification number for the software licensing procedure. Once entered, you need not re-enter the customer_id.

The customer_id is printed next to the codeword on your HP Software Certificate which you receive with the software.

The customer_id must be typed in either using the *-x customer_id=* option or by using the Interactive User Interface.

**default_private_root_tree = /export/private_roots/**

Defines the default location of the private root tree.

Applies to `swinstall`, `swlist` and `swremove`.

**default_shared_root_tree = /export/shared_roots**

Defines the default location of the shared root tree.

Applies to `swinstall`, `swlist` and `swremove`.

**defer_configure = false**

Normally set to "false," `swinstall` will configure the software just installed. If an alternate root directory is specified, configuration is not done, since only hosts that are actually using the software should be configured.

If set to "true," configuration is not performed. This option allows configuration to be manually performed later even when the root directory is `/` and is useful when installing multiple versions.

Note: A multiple version of software will not be configured if another version is already configured by `swinstall`. The `swconfig` command must be run separately.

This option is ineffective if *ANY* of the installed software causes a system reboot. In that case, the specification of *ALL* software installed in the session is appended to the *needs_config* file for configuration after reboot.

Applies to `swinstall`.

**enforce_dependencies = true**

Normally set to "true", dependencies will be enforced. The install, configure and copy commands will not proceed unless necessary dependencies have been selected, or already exist in the proper state ("installed," "configured" or "available", respectively). This prevents unusable software from being installed on the system. It also ensures that depots contain usable sets of software. The remove command also supports this option which means that, by default, dependent software cannot be removed.

If set to "false," dependencies will still be checked, but not enforced. Corequisite dependencies, if not enforced, may keep the software from working properly. Prerequisite dependencies, if not enforced, may cause the installation or configuration to fail.

Applies to `swinstall`, `swcopy`, `swremove`, `swconfig`, `swverify`.

**enforce_dsa = true**

Normally set to "true," installations will not proceed if the disk space required to install the software is more than the available free space. Use this option to allow installation into minfree space, or to attempt the load even though it may fail because the disk reaches its absolute limit.

If set to "false," space checks are still performed but a warning is issued that the system may not be usable if the disk fills past the minfree threshold. An installation will fail if you run out of disk space.

Applies to `swinstall, swcopy, swpackage.`

**enforce_kernbld_failure = true**

Controls whether or not a failure in either of the kernel build steps (*system_prep* and *mk_kernel*) is fatal to the install session. A failure to build a kernel will cause the install process to exit if in non-interactive mode, or to suspend if in an interactive mode.

If set to "false", a failure return from a kernel build process will be ignored, and the install session will proceed. The currently running kernel will remain in place.

**enforce_scripts = true**

Normally set to "true," if a product or fileset *checkinstall* or *checkremove* script fails (that is, returns with exit code 1), none of the filesets in that product will be loaded.

If set to "false," the operation will proceed even though a check script fails. This option can be used to force installation or removal by circumventing check scripts.

Applies to `swinstall, swremove.`

**files =**

When adding or deleting file objects, this option can list the pathnames of those files. There is no supplied default. (File objects being added can also be specified in the given product specification file.)

If there is more than one pathname, they must be separated by whitespace and surrounded by double-quotes.

Applies only to `swmodify.`

**follow_symlinks = false**

Do not follow symbolic links that exist in the packaging source; instead, package them as symlinks.

Applies to `swpackage`.

**include_file_revisions = false**

Normally set to "false," controls whether `swpackage` includes each source file's revision attribute in the product(s) being packaged. Because this operation is very time consuming, the revision attributes are not included by default.

A value of "true" for this keyword causes `swpackage` to execute `what(1)` and possibly `ident(1)` (in that order) to try to determine a file's revision.

Applies to `swpackage`.

**install_cleanup_cmd = /usr/lbin/sw/install_clean**

The script called by the agent to perform release-specific install cleanup steps immediately after the last postinstall script has been run. For an OS update, this script should at least remove commands that were saved by the "install_setup" script.

**install_setup_cmd = /usr/lbin/sw/install_setup**

Defines the script called by the agent to perform release-specific install preparation. For an OS update, this script should at least copy commands needed for the checkinstall, preinstall, and postinstall scripts to a path where they can be accessed while the real commands are being updated.

**kernel_build_cmd = /usr/sbin/mk_kernel**

This is the script called by the agent for kernel building.

**kernel_path = /stand/vmunix**

The path to the system's bootable kernel.

**layout_version = 1.0[0.8]**

Defines the semantics to use when parsing the PSF file. To ensure POSIX 1387.2 semantics, define a layout_version of 1.0.

**level =**

Level designation to list: *depots*, *bundles*, *products*, *subproducts*, *filesets* or *files*.

Applies to `swlist`, `swacl` and `swreg`.

**logfile=/var/adm/sw/<command>.log**

This is the default controller logfile for each command. The agent logfiles are always located relative to the depot or installation, `<depot_path>/swagent.log` and `<root>/var/adm/swagent.log`, respectively. `<root>` is `/` unless an alternate root install was performed.

Applies to all commands except `swlist` and `swacl`.

**logfile=/var/adm/sw/swagentd.log**

This is the default daemon logfile.

**loglevel=x**
**logdetail=false[true]**

Here are the possible combinations of *loglevel* and *logdetail* options:

**Table A-1.**

| Log Level | Log Detail | Information Included |
|---|---|---|
| loglevel=0 | | No information is written to the logfile |
| loglevel=1 | logdetail=false | Only POSIX events are logged. (This is the default.) |
| loglevel=1 | logdetail=true | POSIX detail as above plus task progress messages. (Setting the *loglevel=1* option is not necessary because it is the default.) |
| loglevel=2[1] | logdetail=false | POSIX and file level messages only. Setting the *logdetail=false* option is not necessary. |
| loglevel=2[2] | logdetail=true | All generated information is logged. |

1 Required

2 Setting both the *loglevel=2* and *logdetail=true* options is required. With this combination you may get the same logfile behavior as previous HP-UX 10.x releases.

**log_msgid=0**

Controls whether numeric identification numbers are prepended to logfile messages produced by SD-UX. The default value of 0 indicates no such identifiers are attached. Values of 1-4 indicate that identifiers are attached to messages:

1 applies to ERROR messages only

2 applies to ERROR and WARNING messages

3 applies to ERROR, WARNING, and NOTE messages

4 applies to ERROR, WARNING, NOTE, and certain other logfile messages.

**match_target = false**

If set to "true," forces selection of filesets from the source that match filesets already installed on the target system.

Filesets on the source which specify an installed fileset as an "ancestor" will be selected.

This option overrides any other software selections.

Selections cannot be ambiguous.

Applies to `swinstall` only.

**max_agents = -1**

The maximum number of agents that are permitted to run simultaneously. The value of -1 means there is no limit.

**media_capacity = 1330**

If you are creating a distribution tape, this default specifies the capacity of the tape in one million byte units (not Mbytes). This option is required if the media is not a DDS tape or a disk file. Without this option, `swpackage` sets the size to:

```
tape          1330 megabytes
disk file     free space on the disk up to minfree
```

Applies to `swpackage`.

**mount_all_filesystem = true**

Normally set to "true," the commands automatically try to mount all file systems in the file system table (`/etc/checklist`) at the beginning of the analysis phase and make sure that all those file systems are mounted before proceeding.

When set to "false," no additional file systems are mounted.

Applies only to `swinstall`, `swcopy`, `swverify`, `swremove`, `swconfig`.

**mount_cmd = /sbin/mount**

This is the command called by the agent to mount all file systems.

**objects_to_register =**

Defines the default root selections. If there is more than one root selection listed, they must be separated by spaces.

Applies to `swreg`.

**one_liner = <attributes>**

If there is more than one *attribute*, they must be separated by a space and surrounded by quotes.

```
one_liner="revision size title"
```

You must choose which attributes (that is, revision, size, title, etc.) a default listing of software should use. Note: the *tag* attribute is always displayed for bundles, products, subproducts and filesets; the *path* will always be displayed for files.

Any attributes may be chosen but a particular attribute may not exist for all applicable software classes (bundle/product/subproduct/fileset). For example, the software attribute, *title* is available for bundles, products, subproducts and filesets, but the attribute *architecture* is only available for products.

In the absence of the `-v` or `-a` option, `swlist` will display *one_liner* information for each software object (bundles, products, subproducts and filesets).

Applies only to `swlist`.

**package_in_place = no**

Normally set to "no," a value of "yes" for this keyword causes `swpackage` to build the specified products such that the depot will not actually contain the files that make up a product. Instead, `swpackage` inserts references to the original source files used to build a product. This behavior allows the packaging of products in a development or test environment without consuming the full disk space of copying all the source files into the distribution depot.

Applies to `swpackage`.

**polling_interval = 2**

Normally set to 2. The *polling_interval* option applies only to interactive sessions. It specifies how often each target will be "polled" for status information during the analysis and execution phases. When operating across wide-area networks, the polling interval can be increased (higher number=longer interval between polls) to reduce network overhead.

Applies to `swinstall`, `swcopy` and `swremove`.

**reboot_cmd = /sbin/reboot**

This is the command called by the agent to reboot the system.

**register_new_depot = true**

Normally set to "true," a newly created depot is registered on its host. This allows other commands to automatically "see" this depot.

If set to "false," new depots are not registered. This could allow you to create a "private" depot on which to test, then later register it with `swreg`.

Applies only to `swcopy`.

**register_new_root = true**

Causes roots to be registered during `swinstall`.

**reinstall = false**

Normally set to "false," an existing revision of a fileset that is already installed or available will not be re-installed or re-copied.

If set to "true," the fileset will be re-installed or re-copied.

Applies to `swinstall`, `swcopy`.

**reinstall_files = true**

Normally set to "true," all files within a fileset will be re-installed (overwritten) when that fileset is installed or re-installed (see *reinstall=false* option above).

When set to "false," files that have the same *checksum* (see next option), *size* and *mtime* will not be re-installed. This enhances performance on slower networks.

Applies to `swinstall`, `swcopy`, `swpackage`.

**reinstall_files_use_cksum = true**

Normally set to "true," this option affects the operation when the *reinstall_files* option is set to false. Using the cksum is slower, but is a more robust way to check for files being equivalent. This is the default for `swinstall` and `swcopy`. The default is "false" for `swpackage`, a less reliable, but faster option.

Applies to `swinstall, swcopy, swpackage.`

**remove_empty_depot = true**

When the last product (or bundle) in a depot is removed, the depot itself will be removed.

If set to false, the depot is not removed when the last product (or bundle) in it is removed.

Applies only to `swremove.`

**rpc_binding_info = ncadg_ip_udp:[2121]**

An advanced feature that determines what protocol sequence(s) and endpoint(s) should be used to contact the daemon. This should be consistent among all hosts that work together. This value can be a list of one or more entries of the following form:

- A DCE string binding containing a protocol sequence (for example, *ncacn_ip_tcp*) and, optionally, an endpoint (a port number). The syntax is: `protocol_sequence: [`*endpoint*`]`. If an endpoint is specified, the daemon uses the specified protocol sequence with the specified "well-known" endpoint. If no endpoint is specified, this form has the same effect as the next form.

- The name of a DCE protocol sequence. An entry of this form shows that the named protocol sequence should be used; it does not mean an endpoint. DCE must be installed and the DCE endpoint mapper *rpcd* must be running. It will be used to find the endpoint registered by the daemon.

- The literal string "all." This entry means to use all protocol sequences supported by the Remote Procedure Call (RPC) runtime. It should be the only entry in the list. The *rpcd* must be running.

Applies to all commands except `swpackage.`

**rpc_timeout = 7**

Relative length of the communications timeout. This is a value in the range from 0 to 9 and is interpreted by the RPC runtime. Higher values mean longer times; you may need a higher value for a slow or busy network. Lower values will give faster recognition of attempts to contact hosts that are not running or non-existent.

Applies to all commands except `swpackage.`

**select_local = true**

Normally set to "true," selects the default depot or installation directory of the local host as the target of the command. If the `-t` *target file* is not specified or there were no targets otherwise specified on the command line, the local host is selected. This option applies only to the Command Line User Interface only. The GUI (in the multiple target case) will never automatically select the local system as a target.

Applies to all commands.

The `-t` option only applies to the HP OpenView Software Distributor product and in diskless client installations and removals. **software =**

There is no supplied default. If there is more than one *software_selection*, they must be surrounded by brackets { } or quotes. Software is usually specified in a software_selections input file, as options on the command line or in the GUI or TUI.

Applies to all commands.

**software_view = all_bundles**

Indicates which software view is to be used in the GUI. It can be set to *products*, *all_bundles*, or a *bundle category* tag (shows only bundles of that category). The default view is all_bundles plus products that are not part of a bundle.

Applies to `swcopy`, `swinstall`, `swlist` and `swremove` commands.

**source_directory = :/var/spool/sw**

The default depot location is `/var/spool/sw`. To change this location, use the syntax `host:/path`.

Applies to `swinstall`, `swcopy`, `swpackage`.

**source_file = psf**

This keyword defines the default *package_specification_file* to read as input to the packaging or `swmodify` session. It may be a relative or absolute path.

Applies to `swpackage` and `swmodify`.

**source_tape = :/dev/rmt/0m**

The default tape location, usually the name of an archive device with an optional host (`host:/path`).

Applies to `swinstall`, `swcopy`.

**source_type = directory**

The default source type (choices are *directory*, *tape* or *file*) that points to one of the next three options. The source type derived from the `-s` *source file* option overrides this default.

Applies to `swinstall`, `swcopy`, `swpackage`.

**system_file_path = /stand/system**

The path to the kernel's template file.

**system_prep_cmd = /usr/lbin/sysadm/system_prep**

The kernel build preparation script called by the agent. This script must do any necessary preparation so that control scripts can correctly configure the kernel that is about to be built.

**target_directory = /var/spool/sw**

Defines the default distribution directory in which products will be packaged. The `-d` option overrides this default.

Applies to all commands except `swinstall` and `swconfig`.

**target_tape = /dev/rmt/0m**

Defines the default tape on which products will be packaged. The `-d` options overrides this default.

Applies to `swpackage`.

**target_type = directory**

Defines the type of distribution to create. The recognized types are *directory* and *tape*. Without this option, `swpackage` creates a distribution directory (depot) by default.

Applies to `swpackage`.

**targets =**

There is no supplied default (see *select_local* above). If there is more than one target, they must be surrounded by brackets { } or quotes. Targets are usually specified in a target input file, as options on the command line or in the GUI. Multiple target specification is available only with the HP OpenView Software Distributor product.

Applies to all commands except `swpackage`.

**uncompress_cmd = /usr/bin/uncompress**

This is the command called by the source agent to uncompress files after transmission.

**uncompress_files = false**

When set to "true," files are uncompressed using the current *uncompress_cmd*.

Only one of the *uncompress_files* and *compress_files* options may be set to "true" during a `swpackage` session.

The *uncompress_files* option may not be set to "true" if *package_in_place* is set to true or if the *media_type* is set to "tape."

**use_alternate_source = false**

When the local host begins an analysis or task, the request usually includes information describing the source binding and depot path that the local host should use as the software source.

If "false" (the normal case), the local host will use this information to contact the source.

If "true," the local host will instead use its own configured value.

On the local host, the agent's configured value for *alternate_source* is specified in `host:/path` format. If this value contains only a path component (for example, *alternate_source=:/path*), the agent will apply this path to the file system of its own local host.

If only the host component exists (for example, *alternate_source=host*), the agent will apply the controller-supplied path to this host. If there is no configured value at all for the *alternate_source*, the agent will apply the controller-supplied path to its own local host.

Applies to `swinstall, swcopy.`

**verbose = 1**

By default, the command sends output to `stdout` for task summary messages. Alternatively, the verbose option can be set to 0 for session level messages or (for `swpackage` only) to 2 for file level messages.

Applies to all commands.

**write_remote_files = false**

Normally set to “false,” controls whether `swpackage` will write to a depot that exists on a remote (NFS) file system. By default, files that would be installed, copied or removed on a remote (NFS) file system will be skipped.

If set to “true” and superuser has **NFS root access** on the remote file system, `swpackage` will create or change a depot on that file system.

Applies to `swinstall`, `swcopy`, `swremove`, `swpackage` and `swconfig`.

B

# Control Scripts

SD-UX supports execution of both product and fileset **control scripts**. These shell scripts allow you to perform additional, customized checks and operations as part your regular software management tasks. The `swinstall`, `swconfig`, `swverify` and `swremove` commands can execute one or more of these scripts. Control scripts are usually supplied by the software vendor, but you can also write your own. All these scripts are optional.

*Product level control scripts* are run when any fileset within that product is selected for installation or removal so the activities in product control scripts must pertain to all filesets in that product, but not to any fileset in particular. Actions you want to apply to every fileset in a product should be in the appropriate product level control script.

*Fileset scripts* must pertain only to the installation, configuration, or removal of that fileset, and not to any other fileset or to the parent product.

Control scripts can perform a wide variety of customization and configuration tasks, such as:

- Checking to see if someone is actively using the product and, if so, preventing reinstallation, update or removal.
- Checking the local host system to ensure it is compatible with the software (scripts can check beyond the compatibility enforced by the product's `uname` attributes).
- Removing obsolete files or previously installed versions of the product.
- Creating links to, or additional copies of, files after they have been installed.
- Copying configurable files into place on first-time installation.
- Conditionally copying configurable files into place on later updates.
- Modifying existing configuration files for new features.
- Rebuilding custom versions of configuration files.

- Creating device files or custom programs.
- Killing and/or starting daemons.

**Note** Make sure you specify the path to a shell that is proper for your system. If you get the following message when you execute a script:

```
Cannot execute /var/adm/sw/products/PRODUCT/FILESET/configure.
Bad file number (9).
```

it means the shell in your script has a path that is not correct for your system. (HP-UX 9.0 scripts = *#!/bin/sh* HP-UX 10.0 scripts = *#!/sbin/sh*.)

## Types of Control Scripts

Here are the control scripts that SD-UX supports:

- **Checkinstall Script**

  This script is run by `swinstall` during its Analysis phase to insure that the installation (and configuration) can be attempted. For example, the OS run state, running processes, or other prerequisite conditions beyond dependencies could be checked. It should not change the state of the system.

  A checkinstall script's chief merit is its ability to detect if the system contains a hardware configuration that might lead to catastrophe - an unbootable system or file system corruption - if the installation of the selected software was allowed to proceed. It also acts as the test for conflicts with other software selections or with software already installed.

- **Preinstall Script**

  This script is run by `swinstall` before loading the software files. For example, this script could remove obsolete files, or move an existing file aside during an update.

  This script and the postinstall script are part of `swinstall`'s Load phase. In the Load phase, a product's preinstall script (if any) runs first. Then, for each selected fileset in that product, the fileset's preinstall script runs. The files are then loaded, and the fileset's postinstall script runs. Finally, the product's postinstall script (if any) runs. Then SD loads the next product that has selected filesets and the process repeats for that product.

- **Postinstall Script**

  This script is run by `swinstall` after loading the software files. For example, this script could move a default file into place.

- **Configure Script**

  This script is run by `swinstall` or by `swconfig` to configure the host for the software, or configure the software for host-specific information. For example, this script could change a host's specific configuration file such as `/etc/services`, add the host name or other host resources such as available printers to its own configuration file, or perform compilations.

  Configure scripts are run by `swinstall` for all products (in prerequisite order) *after* the products have completed the Load phase. However, they are only run when installing to a system that will actually be using the software. They are deferred when installing to an alternate root (for example, for diskless or building test file systems) and run instead by the `swconfig` command when the alternate root is now the root of the system using the software.

  The `swconfig` command can also be used to rerun configure scripts that failed during a normal install. A successful execution of the configure step (whether there is a script or not) moves the software from the INSTALLED state to the CONFIGURED or ready-to-use state. Configure scripts (and all others) must be able to be run many times (that is, they must be re-executable).

  Configure scripts are not run for installations to alternate roots.

- **Verify Script**

  Verify scripts are run by the `swverify` command any time after the software has been installed and configured. Like other scripts, they are intended to verify anything that the SD-UX software management tools do not verify by default. For example, this script could check to see that the software is configured properly and that you have a proper license to use it.

- **Unconfigure Script**

  This script is run by `swconfig` or by `swremove` to undo the configuration of the host or the software that was done by the configure script. For example, it could remove its configuration from the `/etc/services` file. (The unconfigure task moves the software from the CONFIGURED state back to the INSTALLED state.)

Only the `swremove` command actually removes software. You should be able to unconfigure and reconfigure using `swconfig`. Unconfigure scripts are not run for removals from alternate roots.

- **Checkremove Scripts**

  The checkremove script is run by `swremove` during the remove Analysis phase to allow any vendor-defined checks before the software is permanently removed. For example, the script could check whether anyone was currently using the software.

- **Preremove Scripts**

  This script is executed just before removing files. It can be destructive to the application because files will be removed next. It could remove files that the postinstall script created.

  This script and the postremove script are part of the Remove phase of `swremove`. Within each product, preremove scripts are run (in the reverse order dictated by any prerequisites), files are removed, then all postremove scripts are run.

- **Postremove Scripts**

  This script is executed just after removing files. It is the companion script to the postinstall script. For example, if this was a patch fileset, then the preinstall script could move the original file aside, and this postremove script could move the original file back if the patch was removed.

- **Other Scripts**

  The software vendor can include other control scripts, such as a subscript that is sourced by the above scripts. The location of the control scripts is passed to all scripts via the {*SW_CONTROL_DIRECTORY*} environment variable.

## Control Script Format

A control script should be a shell script (as opposed to a binary) and written to be interpreted by the Posix.2 shell */sbin/sh* . Korn shell (formerly */bin/ksh*) syntax is acceptable to the Posix.2 shell. A script written for *csh* is not supported.

The script should have a simple header similar to the example below. Included in the header should also be comment lines which state the product and fileset to which the script belongs, the name of the script, the revision string as required by the `what(1)` command, and a simple copyright statement.

```
#! /sbin/sh
########
# Product: <PRODUCT>
# Fileset: <FILESET>
# configure
# @(#) $Revision: 10.0 $
########
#
# (c) Copyright MyCompany, 1994
#
########
```

The following table describes the control script keywords:

**Table B-1. Control Script Keywords**

| Keyword | Value | Example |
|---|---|---|
| checkinstall | path_string, 1024 bytes | /mfg/sd/scripts/checkinstall |
| preinstall | path_string, 1024 bytes | /mfg/sd/scripts/preinstall |
| postinstall | path_string, 1024 bytes | /mfg/sd/scripts/postinstall |
| configure | path_string, 1024 bytes | /mfg/sd/scripts/configure |
| unconfigure | path_string, 1024 bytes | /mfg/sd/scripts/unconfigure |
| verify | path_string, 1024 bytes | /mfg/sd/scripts/verify |
| checkremove | path_string, 1024 bytes | /mfg/sd/scripts/checkremove |
| preremove | path_string, 1024 bytes | /mfg/sd/scripts/preremove |
| postremove | path_string, 1024 bytes | /mfg/sd/scripts/postremove |
| control_file | path_string, 1024 bytes | /mfg/sd/scripts/subscripts |

The value of each keyword is the source filename for the specific control script. `swpackage` will copy the specified control script's filename into the depot's storage directory for the associated product or fileset, using the keyword as the tag of the stored script (for example, "configure").

Additional control scripts or data files can be included with the product or fileset and stored alongside the real control scripts (for example, a subscript called by the supported control scripts, or a data file read by these scripts). These additional scripts are specified using the syntax:

```
PATH[=tag]
```

If the optional *tag* component of the value is not specified, `swpackage` uses the `basename(1)` of the source pathname as the tag for the control script. (Otherwise, the *tag* value is used.)

### Control Script Location on the File System

The checkinstall, preinstall, postinstall, and auxiliary scripts for a fileset will be downloaded to a temporary directory from which they will be invoked:

```
/var/tmp/<CATALOG_DIR>/catalog/<PRODUCT>/<FILESET>/control_script
```

The form of the *<CATALOG_DIR>* is: `aaaa<pid>`, where `<pid>` is the `swinstall` process ID number.

The scripts are delivered to that location from the depot immediately after Product Selection has completed, at the beginning of the Analysis phase and before any system checks have begun. The temporary directory is removed automatically upon exiting `swinstall`.

After successful fileset installation, all other control scripts will be located in the Installed Product Database (IPD). They will be delivered to that location from the depot as part of the installation of the fileset's other files:

```
/var/adm/sw/products/<PRODUCT>/<FILESET>/control_script
```

The location of the IPD is relative to the root directory under which the software installation is done. If the installation is to an alternate root, */mnt/disk2* for example, then the IPD for that software will be under:

```
/mnt/disk2/var/adm/sw/products/<PRODUCT>/<FILESET>
```

**Note** All necessary directories under */var/adm/sw* will be created by the SD-UX process. All files under those directories will be filled by SD-UX initiated processes. Files must never be delivered directly under */var*; it is a private directory.

## General Script Guidelines

Here are some guidelines for writing control scripts:

- All scripts are executed serially and directly impact the total time required to complete an installation, configuration, or removal task. Consider the impact control scripts will have on performance.

- The current working directory in which the agent executes a control script is not defined. Use the environment variables provided by the agent for all pathname references.
- Disk space analysis does not account for files created, copied or removed by control scripts.
- The control scripts you write may be executed several times (for example, configure, then unconfigure, then configure . . . ) so they must be able to support multiple executions.
- You may have to re-execute or debug control scripts, especially when they generate error or warning conditions, so your scripts should be well-written and commented.
- Control script `stdout` and `stderr` are both logged, so output should be restricted to only that information the user requires.

## Environment Variables

All control scripts are invoked as the superuser and executed by the agent process. HP-UX provides the following environment variables for use by a script:

### LANG

Determines the language in which messages are displayed. If *LANG* is not specified or is set to the empty string, a default value of "C" is used.

This variable applies to all SD commands except `swcluster`, `swgettools`, `swpackage`, and `update`.

Note that the language in which the SD agent and daemon log messages are displayed is set by the system configuration variable script, `/etc/rc.config.d/LANG`. For example, `/etc/rc.config.d/LANG` must be set to "LANG=ja_JP.SJIS" or "LANG=ja_JP.eucJP" to make the agent and daemon log messages display in Japanese.

### SW_PATH

The search path for commands. A PATH variable defines the minimum set of commands available for use in a control script (for example, `/sbin:/usr/bin:/usr/ccs/sbin`).

A control script should always set its own PATH variable, and the PATH variable must begin with $SW.PATH. The PATH should be set as follows:

```
PATH=$SW_PATH
export PATH
```

Additional directories, like */usr/local/bin*, can be appended to PATH, but you must make sure that the commands in those directories exist.

## SW_ROOT_DIRECTORY

Defines the root directory in which the session is operating, either "/" or an alternate root directory. This variable tells control scripts the root directory in which the products are installed. A script must use this directory as a prefix to SW_LOCATION to locate the product's installed files.

All control scripts (except for the configure and unconfigure scripts) can be executed during an install or remove task on an alternate root. If the scripts reference any product files, each reference must include the {SW_ROOT_DIRECTORY} in the file pathname.

The scripts may only need to perform actions when installing to (removing from) the primary root directory ("/"). If so, then the SW_ROOT_DIRECTORY can be used to cause a simple exit 0 when the task is operating in an alternate root directory:

```
if test "${SW_ROOT_DIRECTORY}" != "/"
then
     exit 0
fi
```

## SW_LOCATION

Defines the location of the product, which may have been changed from the default **product directory** (if the product is locatable).

When installing to (or removing from) the primary root directory ("/"), this variable is the absolute path to the product directory. For operations on an alternate root directory, the variable must be prefixed by SW_ROOT_DIRECTORY to correctly reference product files.

If a product is not locatable, then the value of SW_LOCATION will always be the default product directory defined when the product is packaged.

## SW_CONTROL_DIRECTORY

Defines the full pathname to the directory containing the script, either a temporary catalog directory, or a directory within in the Installed Products

Database (IPD). This variable tells scripts where other control scripts for the software are located (for example, subscripts).

To source a subscript or reference a data file packaged along with a control script:

```
. ${SW_CONTROL_DIRECTORY}subscript
grep something ${SW_CONTROL_DIRECTORY}datafile
```

### SW_INITIAL_INSTALL

This variable is normally unset. If it is set, the `swinstall` session is being run as the back end of an initial system software installation (that is, "cold" install).

### SW_DEFERRED_KERNBLD

This variable is normally unset. If it is set, the actions necessary for preparing the system file */stand/system* cannot be accomplished from within the postinstall scripts, but instead must be accomplished by the configure scripts. This occurs whenever software is installed to a directory other than `/`, such as for a cluster client system. This variable should be read only by the configure and postinstall scripts of a kernel fileset.

### SW_KERNEL_PATH

The path to the kernel. The default value is */stand/vmunix*.

### SW_SYSTEM_FILE_PATH

The path to the kernel's system file. The default value is */stand/system*.

## Execution of Control Scripts

This section details how each control script is executed.

### Details Common to All Control Scripts

- The agent runs as the superuser, therefore control scripts are always executed as the superuser. Use appropriate caution.
- Control scripts are only executed for software (being) installed, removed or verified in the primary root ("/") or an alternate root directory. Scripts are never executed for software in a depot.
- Each script must set its own `PATH` variable, using SW_PATH.
- Neither `swinstall` nor `swremove` require that the system be shut down. Control scripts must work correctly on both quiet single-user systems and active multi-user systems. They must deal properly with unremovable running programs. They might have to shut down or start up processes that they own themselves to succeed.
- Control scripts can be re-executed. If a script is run more than once, it should produce the same results each time. The second execution should not produce any error messages or leave the system in a state different than before it was run.

  A script should be executable after its fileset was loaded without damaging the new fileset with which it is associated.

  For example, if you must copy a file from under */usr/newconfig* to another location, use the `cpio -p` command to copy it rather than the `cp` command to move it, or check for the absence of the */usr/newconfig* version before attempting the move. (The `cpio(1)` command may be preferred over `cp(1)` because `cpio` copies the mode, owner, and group permissions.)
- Control scripts must exit with a return value of zero (`exit 0`) if no serious errors occur (no ERROR or WARNING messages printed, as described in the "Input and Output" section below.) They must return 1 (`exit 1`) in case of any serious errors, and 2 (`exit 2`) for warnings.

  All messages produced by control scripts are redirected to the agent logfile.
- The set of control scripts executed during a particular phase of a task are always executed in prerequisite order the scripts of each prerequisite product/fileset are executed before the script of the dependent fileset.

## Checkinstall Scripts

- Checkinstall scripts are executed during the Analysis phase of a `swinstall` session. The pathname of the script being executed is:

  `${SW_CONTROL_DIRECTORY}checkinstall`

- A checkinstall script must not modify the system.
- A checkinstall script determines whether the product/fileset can be installed by performing checks beyond those performed by `swinstall`. Example checks include checking to see if the product/fileset is actively in use, or checking that the system run-level is appropriate.
- If the checkinstall script fails, the fileset will not be installed. The interactive interface of `swinstall` will notify you that the checkinstall script has failed. Then you can: diagnose the problem, fix it and re-execute the analysis phase; or unselect the product/fileset. The non-interactive interface tells you about each individual checkinstall failure and the filesets are not installed.
- A checkinstall script is executed for installations into the primary root ("/") or an alternate root. Since most of the actions of this script will involve checking the current conditions of a running system (that is, the primary root), it may not need to perform any actions when the product/fileset is being installed into an alternate root.

## Preinstall Scripts

- Preinstall scripts are executed during the Load phase of a `swinstall` session. The pathname of the script being executed is:

  `${SW_CONTROL_DIRECTORY}preinstall`

- The preinstall script for a product is executed immediately before the fileset's files are installed.
- A preinstall script should perform specific tasks preparatory to the files being installed. The `swinstall` session will proceed with installing the files regardless of the return value from a preinstall script. Example actions include removing obsolete files (in an update scenario).
- A preinstall script is executed for installations into the primary root ("/") or an alternate root. The scope of actions of a preinstall script should be within the product itself (that is, the files within the product's directory).

## Postinstall Scripts

- Postinstall scripts are executed during the Load phase of a `swinstall` session. The pathname of the script being executed is:

  ```
  ${SW_CONTROL_DIRECTORY}postinstall
  ```

- The postinstall script for a product is executed immediately after the fileset's files are installed.
- A postinstall script should perform specific tasks related to the files just installed. The `swinstall` session will proceed with the remainder of the session (for example, configuration) regardless of the return value from a postinstall script. Example actions include adding a kernel driver to the system file or moving a file from under */usr/newconfig* to its correct place in the file system.
- A postinstall script is executed for installations into the primary root ("/") or an alternate root. The scope of actions of a postinstall script should be within the product itself (that is, the files within the product's directory).
- The customization or configuration tasks that must be performed to enable the product/fileset for general use should not be done in the postinstall script, but the configure script (described below).

## Configure Scripts

- Configure scripts are executed during the Configuration phase of a `swinstall` session. SD expects configure scripts at system startup if the `swinstall` session triggers a system reboot. The `swconfig` command can also execute configure scripts. The pathname of the script being executed is:

  ```
  ${SW_CONTROL_DIRECTORY}configure
  ```

- A configure script is only executed for installations into the primary root ("/"). If you choose to defer configuration in the `swinstall` session, then the configure script will be executed by a `swconfig` session at some time after the installation completes.
- A configure script is usually executed only when the product/fileset is in the INSTALLED state.

- A configure script is the primary way to move a product/fileset from the INSTALLED state to the CONFIGURED state. The script should perform all (or most of) the activities needed to enable the product/fileset for use.
- When an existing version of a product is updated to a new version, the configure script(s) for the new version must perform any unconfigurations-configurations of the old version that are necessary to properly configure the new version. The unconfigure script(s) for the old version are not executed.
- Configure scripts are for architecture dependent actions because they will always be run on the architecture of the install target.

## Unconfigure Scripts

- Unconfigure scripts are executed during the Unconfiguration-Configuration phase of a `swremove` session. They can also be executed by the `swconfig` command. The pathname of the script being executed is:

  ```
  ${SW_CONTROL_DIRECTORY}unconfigure
  ```
- An unconfigure script is executed only for software installed into the primary root ("/").
- An unconfigure script is re-executed even when the product/fileset is in the CONFIGURED state.
- An unconfigure script is the primary way to move a product/fileset from the CONFIGURED state back to the INSTALLED state. The script should perform all (or most of) the activities needed to disable the product/fileset for use.
- An unconfigure script must undo all configuration tasks performed by its companion configure script. The user should be able to configure, unconfigure, configure, . . . etc. an installed product/fileset and always end up with the same configured result.

## Verify Scripts

- Verify scripts are executed by the `swverify` command. The pathname of the script being executed is:

  ```
  ${SW_CONTROL_DIRECTORY}verify
  ```
- A verify script must not modify the system.

- A verify script is the primary way to check the configuration tasks performed by a configure script for correctness and completeness.
- A verify script is executed for installations into the primary root ("/") or an alternate root. Since most of the actions of this script will involve checking the current conditions of a CONFIGURED product/fileset (in the primary root), it may not need to perform any actions for a product/fileset installed into an alternate root directory.

## Checkremove Scripts

- Checkremove scripts are executed during the Analysis phase of a `swremove` session. The pathname of the script being executed is:

  ```
  ${SW_CONTROL_DIRECTORY}checkremove
  ```

- A checkremove script must not modify the system.
- A checkremove script determines whether the product/fileset can be removed by performing checks beyond those performed by `swremove`. Example checks include checking to see if the product/fileset is actively in use.
- If the checkremove script fails, no filesets in the product will be removed. The GUI/TUI interface of `swremove` notifies you that the checkremove script has failed. You can then: diagnose the problem, fix it, and re-execute the analysis phase; unselect the target system(s) in question; or unselect the product/fileset. The command line interface notifies you for each individual checkremove failure, and no filesets in that product are removed.
- A checkremove script is executed for installations into the primary root ("/") or an alternate root. Since most of the actions of this script will involve checking the current conditions of a running system (that is, the primary root), it may not need to perform any actions when the product/fileset is being removed from an alternate root.

## Preremove Scripts

- Preremove scripts are executed during the Remove phase of a `swremove` session. The pathname of the script being executed is:

  ```
  ${SW_CONTROL_DIRECTORY}preremove
  ```

- All preremove scripts for a product are executed immediately before the product's files are removed.

- A preremove script should perform specific tasks preparatory to the files being removed. The `swremove` session will proceed with removing the files regardless of the return value from a preremove script. Example actions include removing files created in the postinstall script.
- A preremove script is executed for installations into the primary root ("/") or an alternate root. The scope of actions of a preremove script should be within the product itself (that is, the files within the product's directory).
- The de-customization or unconfiguration-configuration tasks which must be performed to disable the product/fileset for general use must not be done in a preremove script, instead they should be done in an unconfigure script (described above).

## Postremove Scripts

- **Postremove scripts** are executed during the remove phase of a `swremove` session. The pathname of the script being executed is:

  `${SW_CONTROL_DIRECTORY}postremove`

- All postremove scripts for a product are executed immediately after the product's fileset files are removed.
- A postremove script should perform specific tasks related to the files just removed. The `swremove` session will proceed with the remainder of the session regardless of the return value from a postremove script. Example actions include:
  - removing any files still remaining after preremove and the `swremove` file removal have completed.
  - removal of directories wholly owned by the fileset and which have been emptied by the file removal.
- A postremove script is executed for installations into the primary root ("/") and an alternate root. The scope of actions of a postremove script should be within the product itself (that is, the files within the product's directory).
- The de-customization or unconfiguration-configuration tasks which must be performed to disable the product/fileset for general use should not be done in the postremove script, instead they should be done in the unconfigure script (described above).

## Execution of Other Commands by Control Scripts

Every command executed by a control script is a potential source of failure because the command may not exist on the target system. Your script can use any command conditionally, if it checks first for its existence and executability, and if it does not fail when the command is unavailable.

- If the target system(s) conform with the POSIX 1003.2 Shells and Utilities standard, then the Execution Environment Utilities of this standard will also be available.
- If a fileset has a prerequisite dependency on another product/fileset, then most of the control scripts for the dependent fileset can use the commands of the required product/fileset, if the $ROOT_DIRECTORY is `/`. (All commands perform their tasks in prerequisite order).
- Commands should be referenced relative to the path components specified in the `PATH` variable. (See the discussion of `PATH` and the `SW_PATH` environment variable above.)

## Input To and Output From Control Scripts

- Control scripts must not be interactive. This includes messages such as, `Press return to continue`. All control scripts are executed by the agent on the target systems. No method of input to control scripts is supported.
- Control scripts must write messages for error and warning conditions to `stderr` (echo &>2), and write all other messages to stdout. Control scripts must not write directly to `/dev/console` or attempt any other method of writing directly to the display.

  The `stdout` and `stderr` from a control script is redirected by the agent to the log file (*var/adm/sw/swagent.log*) within the primary or alternate root directory in which the task is being performed.

  For interactive `swinstall` and `swremove` sessions, you can display and browse this logfile.
- Only minimal, essential information should be emitted by control scripts. Ideally, no output is emitted if the script successfully performs all of its actions.

- In the agent logfile, the execution of each control script is prefaced by a "begin execution" message:

```
* Running "checkinstall" script for product "PRODUCT".
* Running "checkinstall" script for fileset "PRODUCT.FILESET".
```

  Any messages generated by the script will follow. If the script returns a value other than 0 (SUCCESS), then a concluding message is written:

```
ERROR:   The "unconfigure" script for "PRODUCT.FILESET" failed (exit
         code "1"). The script location was
         "/var/adm/sw/products/PRODUCT/FILESET/unconfigure".
       * This script had errors but the execution of this product
         will still proceed.  Check the above output from the script
         for further details.

WARNING: The "unconfigure" script for "PRODUCT.FILESET" failed (exit
         code "2"). The script location was
         "/var/adm/sw/products/PRODUCT/FILESET/unconfigure".
       * This script had warnings but the execution of this product
         will still proceed.  Check the above output from the script
         for further details.
```

- The messages written by a control script must conform to the following format conventions whenever possible.

  1. Never emit blank lines.
  2. All output lines must have one of these forms:

     ```
     ERROR:   text
     WARNING: text
     NOTE:    text
     blank    text
     ```

     In each case, the keyword must begin in column 1, and the *text* must begin in column 10 (indented nine blanks).
  3. Choose the keyword (`ERROR`, `WARNING`, `NOTE`, or blank) as follows:

| | |
|---|---|
| `ERROR:` | Something happened that must grab your attention. Cannot proceed, or need corrective action (to be taken later). |
| `WARNING:` | Can continue, but it's important that you know something went wrong or requires attention. |
| `NOTE:` | Something out of the ordinary or worth special attention; not just a status message. |

| | |
|---|---|
| blank | Generic progress and status messages (keep them to a necessary minimum). |

Do not start a line with an asterisk (*) character. This is reserved for operational messages printed by the agent so they can be easily distinguished from other messages.

4. If the message text requires more than one, 72-character line, break it into several 72-character lines. Indent all lines after the first. For example:

```
NOTE:    To install your new graphics package, it was necessary to turn
         on the lights in the next room. Please turn them off when you leave.
```

5. Do not use tab characters in any messages.

- Scripts execute other commands which may unexpectedly fail and emit output not in the above format. Wherever you suspect a failure is possible or likely (and it is reasonable to do so) redirect the standard output or error of the executed command to *</dev/null* or to a temporary file. Then emit a proper-format message based on the return code or on output from the command. For example:

```
if /bin/grep bletch /etc/bagel 2 &>/dev/null
then echo "ERROR: Cannot find bletch in /etc/bagel." &>2
fi
```

- The following conventions will help a control script's messages have a similar look and feel to the messages generated by the agent (and the commands themselves).

1. Use full sentences wherever possible. Avoid terseness.
2. Start sentences and phrases with a capital letter and end with a period.
3. Put two blanks after a period; one after colons, semicolons, and commas.
4. Uppercase first letters of phrases after colons. (This helps break up the message into digestible "bites" of information.)
5. Surround product, fileset, directory, and file names, and other variable-valued strings with quotes. For example:

```
echo "ERROR:  Cannot open file \"$file\"." &>2
```

6. Speak in present tense. Avoid "would", "will", and so forth. Also avoid past tense except where necessary.

7. Use “cannot” rather than “can’t”, “could not”, “couldn’t”, “unable to”, “failed to”, and similar phrases.

## File Management by Control Scripts

- If any files in the previous revision of a product have changed names or became obsolete, a product/fileset preinstall or postinstall script in the new revision of the product must remove the old files. The agent does not remove the files in an existing product/fileset before updating it to a newer revision.

**Note** It is necessary to perform the cleanup task of any previous revision that can be updated to the new revision. Sometimes this is more than just the previous revision.

- All files created by a preinstall, postinstall, or configure script must be removed by a companion postremove, preremove or unconfigure script.

  Files created by scripts are not known by the `swremove` command, and will not get removed when it removes those files installed by `swinstall`.

## Testing Control Scripts

The following testing suggestions do not cover all test scenarios. There may still be problems with a control script even after doing this testing. For example, you may test installing/removing individual filesets. But there might be some interactions that are discovered only after all the filesets are installed on or removed from the system.

Similarly, you may test the control scripts on a fully loaded system and miss a problem when you execute a command in your script that is not part of the base (or core) system. If your target system does not contain the particular command, your script may fail.

## Testing Installation Scripts

For checkinstall, preinstall, and postinstall scripts you should perform at least these tests. All tests can be performed on the local system (that is, by doing local installs).

1. The basic test:

   a. Run `swinstall` to install the full product (that is, all the filesets). To avoid testing the configure script(s), either do not include any in the product, or set the *defer_configure* option to "true."

   b. After the installation completes, check the *<${SW_ROOT_DIRECTORY}var/adm/sw/swagent.log* file for any problems, either in the scripts or the format/contents of the messages generated by the scripts.

   c. Study the resulting file system to see if the scripts performed the expected actions.

   d. Re-run the test by re-installing the same product.

2. If you want to avoid the time spent loading files, then set the *reinstall_files* option to "false" and the *reinstall_files_use_cksum* option to "false."
3. If a previous version of the product can be updated to this version, then re-run the test by updating this product where the previous version has been installed.
4. If your checkinstall script can generate ERROR or WARNING conditions based on the current activity or configuration of the target system, then enable those conditions to ensure that the checkinstall script correctly detects them.
5. Re-run the test by installing into an alternate root directory (`swinstall -r`) instead of the primary root directory ("/"). Make sure that the scripts perform all of their operations (if any) within the alternate root directory. (This verifies the correct use of ${SW_ROOT_DIRECTORY} by your scripts.)
6. If your product is locatable (that is, it can be installed into a different location), then re-run the tests by installing the product into a different location (`swinstall` *product*`:new_location`). Make sure that the scripts perform all of their operations in the new location, and not the default location. (This verifies the correct use of $SW_LOCATION by your scripts.)
7. If you have a complex script, run additional tests for your product that you feel will give you confidence your product has been installed correctly on the

system. For example, only install certain subsets of your product instead of the full product.

## Testing Configuration Scripts

For configure, verify, and unconfigure scripts you should perform at least these tests. All tests can be performed on the local system (that is, by doing local installs).

1. Run `swinstall` to install the full product (that is, all the filesets). Let the installation process perform the configuration task (and run your configure script(s)).
   a. After the installation and configuration completes, check the *${SW_ROOT_DIRECTORY}var/adm/sw/swagent.log* file for any problems, either in the configure script or the format/contents of the messages generated by it.
   b. Study the resulting file system to see if the configure script performed the expected actions.
   c. Test the product itself to see if the necessary configuration tasks were performed such that the product is ready to use.
2. Run `swremove` to removed the configured product.
   a. After the unconfiguration and removal completes, check the *${SW_ROOT_DIRECTORY}var/adm/sw/swagent.log* file for any problems, either in the unconfigure script or the format/contents of the messages generated by it.
   b. Study the resulting file system to see if the unconfigure script performed the expected "undo" actions.
3. Run `swinstall` to install the full product again. Set the *defer_configure* option to "false" to avoid executing the configure scripts.
   a. After the installation completes, run `swconfig` to configure your product.
   b. Study the resulting file system to see if the configure script performed the expected actions.
   c. Test the product itself to see if the necessary configuration tasks were performed such that the product is ready to use.
   d. Now run `swconfig -u` to unconfigure your product.

   e. Study the resulting file system to see if the unconfigure script performed the expected “undo” actions.

   f. Run `swconfig` again to re-configure your product.

   g. Study the resulting file system to see if the configure script performed the expected actions.

4. Run `swverify` to execute the verify script(s).

   a. After the verification completes, check the *${SW_ROOT_DIRECTORY}var/adm/sw/swagent.log* file for any problems, either in the verify script or the format/contents of the messages generated by it.

5. If a previous version of the product can be updated to this version, then re-run the first test by updating this product to a system where the previous version has been installed and configured.

6. Note that configure and unconfigure scripts are never run unless the `${SW_ROOT_DIRECTORY}` is `/`. However, verify scripts are run in both cases.

7. If your product is locatable (that is, it can be installed into a different location), then re-run the tests by installing and configuring the product in a different location. Make sure that the scripts perform all their operations in the new location, and not the default location. (This verifies the correct use of `$SW_LOCATION` by your scripts.)

8. If you have a complex script, run additional tests for your product that you feel will give you confidence your product has been installed correctly on the system. For example, only install certain subsets of your product instead of the full product.

## Testing Removal Scripts

For checkremove, preremove, and postremove scripts you should perform at least these tests. All tests can be performed on the local system (that is, by doing local installs). There is no value gained by testing your scripts by installing to remote target systems.

1. Run `swinstall` to install the full product (that is, all the filesets). Avoid configuration by setting the `defer_configure` option to `false`.

   a. Run `swremove` to removed the unconfigured product.

   b. After the removal completes, check the *${SW_ROOT_DIRECTORY}var/adm/sw/swagent.log* file for any problems, either in the removal scripts or the format/contents of the messages generated by the scripts.

   c. Study the resulting file system to see if the removal scripts performed the expected actions.

2. Run `swinstall` to install the full product (that is, all of the filesets). Let the installation process perform the configuration task (and run your configure script(s)).

   a. Run `swremove` to removed the configured product.

   b. After the unconfiguration and removal completes, check the *${SW_ROOT_DIRECTORY}var/adm/sw/swagent.log* file for any problems, either in the removal scripts or the format/contents of the messages generated by the scripts.

   c. Study the resulting file system to see if the removal scripts performed the expected actions.

3. If your checkremove script can generate ERROR or WARNING conditions based on the current activity or configuration of the target system, then enable those conditions to ensure that the checkremove script correctly detects them.

4. Re-run the first test by installing into an alternate root directory (`swinstall -r`) instead of the primary root directory ("/"). Make sure that the scripts perform all of their operations (if any) within the alternate root directory. (This verifies the correct use of `${SW_ROOT_DIRECTORY}` by your scripts.)

5. If your product is locatable (that is, it can be installed into a different location), then re-run the tests by installing the product into a different location. When removing the product, make sure that the removal scripts perform all of their operations in the new location, and not the default location. (This verifies the correct use of `$SW_LOCATION` by your scripts.)

6. If you have a complex script, run additional tests for your product that you feel will give you confidence your product has been installed correctly on the system. For example, only install certain subsets of your product instead of the full product, then perform the remove operations. (Or only remove subsets of the fully installed product.)

C

# Replacing or Updating SD-UX

*Because you do not yet have the new SW-DIST product for HP-UX Version 10.20 on your system (or if SW-DIST is damaged, corrupted or missing) you must get the new SD-UX 10.20 tools to install any 10.20 software packaged with SD.*

## Updating SD-UX Before Installing/Updating Applications

Before you can update to HP-UX 10.20, you *must* extract the new version of SD-UX from the 10.20 tape, CD, or software depot from which you plan to update your system.

**Caution** Do NOT use any previous versions of `swinstall` or `swcluster` to update the system to 10.20. The update will fail.

### Procedure

To update SD, you must first load the `swgettools` command onto your system, and then use `swgettools` to get the new version of SD-UX.

The `swgettools` command needs a temporary directory with at least 11 Mb of free space. If there is not enough space in the temporary directory `swgettools` will fail. By default, `swgettools` uses the `/var/tmp` directory. Use the `bdf /var/tmp` command to determine if `/var/tmp` has adequate space.

If you do not have 11 MB free in `/var/tmp`, you must tell `swgettools` to use a different temporary directory by using the `-t` *dir_path* command-line option.

The `swgettools` command is shipped in the `catalog/SW-GETTOOLS/pfiles` directory. Depending on whether the 10.20 software is on CD, tape, or a remote system in a software depot, use `cp`, `tar`, or `rcp`, respectively, to load swgettools onto your system. See the section “SW-DIST Installation Examples” below for more examples and other options.

After loading the swgettools utility, you can use it to update your system to 10.20.

---

**Caution** Do NOT reboot your system after running swgettools and before you have run swinstall or swcluster to update HP-UX.

If you do have to reboot, you must run swgettools again before you update HP-UX.

---

## SW-DIST Installation Examples

### From CD-ROM

To install the new SW-DIST product from CD-ROM at /mount/cdrom_depot:

```
cp /mount/cdrom_depot/catalog/SW-GETTOOLS/pfiles/swgettools /var/tmp

/var/tmp/swgettools -s /mount/cdrom_depot
```

### From Tape

To install the new SW-DIST product from tape at /dev/rmt/0m:

```
cd /var/tmp

tar -xvf /dev/rmt/0m catalog/SW-GETTOOLS/pfiles/swgettools

cp /var/tmp/catalog/SW-GETTOOLS/pfiles/swgettools /var/tmp/swgettools

rm -rf /var/tmp/catalog

/var/tmp/swgettools -s /dev/rmt/0m
```

### From Remote Depot

To install the new SW-DIST from a remote depot on system swperf at /var/spool/sw, enter the following:

```
rcp swperf:/var/spool/sw/catalog/SW-GETTOOLS/pfiles/swgettools  /var/tmp

/var/tmp/swgettools -s swperf:/var/spool/sw
```

## Updating SD Without Root Access to the Remote Depot

You may use the procedure in Option 1 below. If you are a system administrator, instruct your users to follow Option 2.

Both of the following options assume your users will run `swgettools` specifying a source depot on a remote server, and that you do not want to grant them `rcp` (`.rhosts`) access as root to the server.

### Option 1:

1. Copy the `swgettools` script file and the `swagent.Z` file from the tape or CD (in the `catalog/SW-GETTOOLS/pfiles` directory) to a location that your users have FTP access to.
2. Tell the user to:
   a. FTP the two files into the `/var/tmp` directory on the system to be updated.
   b. Use `chmod +x` to make the swgettools script executable.
   c. Run swgettools and specify the remote depot location with the `-s` option (and, if necessary, `-t` to specify a temporary directory other than `/var/tmp`).

### Option 2:

Users can use the SD `swcopy` command to copy the `SW-GETTOOLS` product from a registered remote source depot to a local depot prior to extracting the files. The remote source depot can be either a CD-ROM or a disk depot.

To copy the `SW-GETTOOLS` product from the remote CD-ROM depot located at `swperf:/mount/cdrom` to a local depot in `/tmp/depot`:

```
swcopy -s swperf:/mount/cdrom SW-GETTOOLS @ /tmp/depot
```

Then copy the swgettools script and the `swagent.Z` file to the `/var/tmp` directory:

```
cp /tmp/depot/catalog/SW-GETTOOLS/pfiles/sw* /var/tmp
```

Execute the `swgettools` script specifying the remote depot to update the `SW-DIST` product:

```
/var/tmp/swgettools -s swperf:/mount/cdrom
```

**Note** If you will use a temporary directory other then `/var/tmp`, then `cp` the `swgettools` script and the `swagent.Z` file to the directory you will use and specify the directory location on the `swgettools` command line using the `-t` *dir_path* command-line option.

### Example:

```
cp /tmp/depot/catalog/SW-GETTOOLS/pfiles/sw* /usr/tmp

/usr/tmp/swgettools -s swperf:/mount/cdrom -t /usr/tmp
```

### For More Information

Consult the *swgettools*(1M) man page (on a 10.10 or later system) for assistance with the following:

- If you encounter an error during the execution of the `swgettools` script.
  *OR*
- If you want to see other `swgettools` examples.

# Glossary

The following terms, names, and acronyms are used in the descriptions of the HP software management commands.

**Access Control Lists (ACL)**
A structure, attached to a software object, that defines access permissions for multiple users and groups. It extends the security concepts defined by the HP-UX file system mode bits by allowing specification of the access rights of many individuals and groups instead of just one of each.

**Agent**
The agent (`swagent`) runs on the local host. It services all selection, analysis, execution and status requests.

**Alternate Depot Directory**
SD-UX allows you to change the default location of a depot directory.

**Alternate Root Directory**
SD-UX allows you to change the location of a root directory to a directory that may eventually be the root of another system.

**Analysis, Analysis Phase**
The second phase of a software installation, copy, or remove operation, during which the host executes a series of checks to determine if the selected products can be installed, copied or removed on the host. The checks include the execution of check scripts and DSA (disk space analysis).

**Ancestor**
An "attribute" which signifies the name of a previous version fileset used in the "Match-What-Target-Has" functionality. If the *match_target* option is set to true, the system will match the ancestor fileset name to the new fileset name.

**Architecture**
This keyword defines the “architecture” attribute for the product object. It refers to the operating system platform on which the product runs.

**Attributes**
Information describing a software object’s characteristics. For example, product attributes include revision number, tag (name), and contents (list of filesets). Fileset attributes include tag, revision, kernal, and reboot. File attributes include mode, owner, and group. An essential part of the PSF, attributes include such information as the product’s short name or tag, a one-line full name title or a one paragraph description of the object. Other attributes include a multi-paragraph README file, a copyright information statement and others.

**Building Phase**
Packaging the source files and information into a product, and creating/merging the product into the destination depot/media.

**Bundles**
A collection of filesets that are encapsulated for a specific purpose. By specifying a bundle, all filesets under that bundle are automatically included in the operation.

**Catalog Files**
This is the area within a depot that contains all the information needed by SD-UX to understand the organization and contents of the products stored in the depot. It includes a global INDEX file and a directory of information for each product version in the depot. It is sometimes referred to as the catalog directory.

**Category**
This keyword defines the “category” attribute for the product object. It refers to the type of software being packaged.

**CD-ROM Media**
Compact Disc-Read Only Memory. One type of media. Essentially, it is a depot that resides on a CD-ROM.

**CD-ROMs**
See CD-ROM Media.

**Checkinstall Script**

An optional, vendor-supplied script associated with a product or fileset that is executed during the `swinstall` analysis phase. The result returned by the script determines if the fileset can be installed or updated.

**Checkremove Scripts**

An optional, vendor-supplied script associated with a fileset that is executed during the `swremove` analysis phase. The result returned by the script determines if the fileset can be removed.

**Clients**

Usually refers to diskless server computer. SD-UX is often used from a diskless server to install software.

**Cluster Server**

Provides facilities needed for clients to boot over the network from kernels residing in the server's file system. The server also provides each client's root (/) file system.

**Compatibility Filtering**

The capability in `swinstall` to filter the software available from a source according to the host's `uname` attributes. Software products are created to run on specific computer hardware and operating systems. Many versions of the same products may exist, each of which runs on a different combination of computer hardware and operating system. The default condition for compatibility filtering in `swinstall` is to NOT allow selection and installation of incompatible software.

**Compatible Software**

A software product that will operate on a given hardware system. Software that passes compatibility filtering for a local host. See Incompatible Software.

**Configure Script**

An optional, vendor-supplied script of `checkinstall` associated with a fileset that is executed by `swinstall` (or `swconfig`) after the installation of filesets is complete.

**Container ACL Templates**

A special ACL (*global_soc_template*) that is used to create initial ACLs for depot and roots.

**Controller**
The controller is the component of the SD-UX software management command that is invoked by the user on the local host.

**Control Scripts**
Optional, vendor- or administrator-supplied scripts that are run during `swinstall` and `swremove`. Includes checkinstall, preinstall, postinstall and configure scripts for `swinstall`; the checkremove, unconfigure, preremove, and postremove scripts for `swremove`; and the verify script for `swverify`.

**Copyright**
This keyword defines the "copyright" attribute for the destination depot (media) being created/modified by `swpackage`. It refers to the copyright information that pertains to the software product.

**Corequisites**
A corequisite dependency requires another software object to be installed in order for the dependent fileset to be usable. See Dependency.

**Critical Filesets**
Critical filesets contain software that is critical to the correct operation of the host. Critical filesets are those with the reboot and/or kernel fileset flags. During the load phase, critical filesets are loaded and customized before other filesets.

**Daemon/agent**
See agent or `swagentd`.

**Defaults File**
The file */var/adm/sw/defaults* (system level defaults) or *$HOME/.sw/defaults* (user level defaults), which contains the default options and operands for each SD-UX software management command.

**Default Option Value**
The changeable values contained in the default option file (above) which are modified with the syntax `swcommand.option=value`.

**Dependent**
A relationship between a fileset and another software object in which the fileset depends on the other object in a specific manner. Also used to identify the software object upon which the dependency exists: for example, if fileset A depends on fileset B, then B is a dependent or dependency of A. SD-UX supports *corequisite* and *prerequisite* dependencies.

**Dependency**
See **Dependent**

**Depot**
A repository of software products and a catalog organized such that the SD-UX software management commands can use it as a source. The contents of a depot reside in a directory structure with a single, common root. A depot can exist as a directory tree on a SD-UX file system or on CD-ROM media, and it can exist as a `tar` archive on a serial media. All depots share a single logical format, independent of the type of media on which the depot resides.

**Depot Source**
A depot that is available for use as a source of software products. A depot source in directory format is identified by the path to a directory where a CD-ROM containing a depot is mounted, or the path to the root directory of a depot in the file system. A depot source in `tar` format is identified by the path to a tape device containing a tape media, or the path to a regular file containing a depot in a `tar` archive. A depot source in directory format can be remotely accessed as a network source through a `swagentd` server.

**Developer Host**
Where software application files are placed for further integration and preparation for distribution. Developer hosts assemble, organize and create product tapes or depots.

**Description**
Vendor-defined descriptive attribute for products and filesets. A paragraph description of the product or fileset.

**Details Dialog**
The Details Dialog allows you to obtain more information regarding the specific process and monitor its progress.

**Directory**
This keyword defines the "directory" attribute for the product object. The default, absolute pathname to the directory in which the product will be installed. Defines the root directory in which the product files are contained.

**Directory Depot**
The directory on a target host where the depot is located. The default is */var/spool/sw*.

**Disk Space Analysis (DSA)**
This process determines if a host's available disk space is likely to be sufficient for the selected products to be installed.

**Downdating**
Overwriting an installed version of software with an older version.

**End**
Ends the depot specification, no value is required. This keyword is optional.

**Filesets**
A collection of files. The software object upon which most SD-UX software management operations are performed.

**GUI**
Graphical User Interface. The OSF/Motif TM user interface provided with the `swinstall`, `swcopy` and `swremove` commands (compare to the CLUI or TUI).

**Host**
A computer system upon which SD-UX software management operations are performed. See local host.

**Host ACL**
The ACL that is attached to, and controls access to, the host object.

**Incompatible Software**
Software products are created to run on specific computer hardware and operating systems. Many versions of the same products may exist, each of which runs on a different combination of computer hardware and operating system. Incompatible software does not operate on the host(s) because of the host's computer hardware or operating system. The default condition in `swinstall` is to dis-allow selection and installation of incompatible software.

**INDEX**
An INDEX file defines attribute and organizational information about an object (for example, depot, product, or fileset). INDEX files exist in the depot catalog and the Installed Products Database to described their respective contents.

**INFO**
An INFO file provides information about the files contained within a fileset. This information includes type, mode, ownership, checksum, size, and pathname attributes. INFO files exist in the depot catalog and the Installed Products Database to described the files contained in each existing fileset.

**Input Files**

Defaults files, option files, software_selection files, target_host files and session_files that modify and control the behavior of the SD-UX software management commands.

**Installed Products**

A product that has been installed on a host so that its files can be used by end-users. Contrast with a product residing in a depot on a host's file system, sometimes referred to as an "available product."

**Installed Products Database (IPD)**

Describes the products that are installed on any given host (or within an alternate root). Installed product information is created by `swinstall`, and managed by `swremove`. The contents of an Installed Product Database reside in a directory structure with a single common root.

**Instance_ID**

A product attribute in the Installed Products Database (IPD) that allows the vendor to uniquely identify products with the same tag (name) or revision.

**IPD**

The Installed Products Database.

**Is_Locatable**

This keyword defines the "is_locatable" attribute for the product object. Defines whether a product can be installed to an alternate product directory or not. If specified, the attribute is set to a value of TRUE. If not specified, the attribute is assigned a value of FALSE.

**Kernel Filesets**

A fileset that contains files used to generate the operating system kernel. During the `swinstall` load phase, kernel filesets are loaded and customized before other filesets.

**Keyword**

A word (or statement) that tells `swpackage` about the structure or content of the software objects being packaged by the user. Packaging information is input to `swpackage` using a Product Specification File.

**Load, Load Phase**

The third phase of a software installation or copy operation; when `swinstall` and `swcopy` load product files on to the host; and when `swinstall` performs product-specific customizations.

**Local Host**

The host on which an SD-UX software management commands are being executed. Equivalent to the administrative host.

**Locatable Products**

A product that can be relocated to an alternate product directory when it is installed. (If a product is not locatable, then it must always be installed within the defined product directory.)

**Logging**

Each of the SD-UX software management commands record their actions in log files (the `swlist` command is an exception). The default location for the various log files is `/var/adm/sw/<command>.log`.

**Machine_Type**

This keyword defines the "machine_type" attribute for the product object. The machine type(s) on which the product will run. (If not specified, the keyword is assigned a wildcard value of "*", meaning it will run on all machines.) If there are multiple machine platforms, you must separate each machine designation with a `|` (vertical bar).

**Make Tape Phase**

In packaging software to a distribution tape, this phase actually copies the contents of the temporary depot to the tape.

**Media**

See Software Media.

**Network Source**

A `swagentd` daemon that makes products in a local depot available for `swinstall` and `swcopy` access. There can be multiple network sources from a single host, each one a different depot served by that host's single `swagentd` daemon. A network source is identified by the host name and depot directory.

**Nodes**

Another name for client host. See Client.

**Number**

This keyword defines the "number" attribute for the destination depot (media) being created/modified by `swpackage`. The part or manufacturing number of the distribution media (CD or tape depot).

**Package Selection Phase**
In `swpackage`, reading the *product_specification_file* to determine the product, subproduct and fileset structure; the files contained in each fileset and the attributes associated with these objects.

**Postinstall Script**
An optional, vendor-supplied script associated with a fileset that is executed by `swinstall` after the corresponding fileset has been installed or updated.

**Postremove Scripts**
An optional, vendor-supplied script associated with a fileset that is executed by `swremove` after the corresponding fileset has been removed.

**Prerequisites**
A prerequisite dependency requires another software object to be installed (configured) in order for the first fileset to be installed (configured). See Dependency.

**Preinstall Script**
An optional, vendor-supplied script associated with a fileset that is executed by `swinstall` before installing or updating the fileset.

**Preremove Scripts**
An optional, vendor-supplied script associated with a fileset that is executed by `swremove` before removing the fileset.

**Principal**
In SD Security, the user (or host system, for agents making RPCs) who is originating the call.

**Private Roots**
Directories on diskless servers that contain the private files of the diskless clients.

**Products**
Collections of subproducts and filesets. The SD-UX software object that directly relates to the software purchased by the user.

**Product ACL Templates**
In software security, the ACL used to initialize the ACLs protecting new products on depots that are created by the host.

**Product Directory**

The root directory of a product object, in which (nearly) all its files are contained. A user can change (relocate) the default product directory when installing a locatable product.

**Product Specification File (PSF)**

The input file used to define the structure and attributes of the products to be packaged by `swpackage`.

**Product Versions**

A depot can contain multiple versions of a product. Product versions have the same tag attribute, but different version attributes. See Multiple Version. An IPD supports multiple installed versions of a product. Installed product versions have the same tag attribute, but different version attributes and/or a different product directory.

**Push**

A "push" installation means that the administrator can, from a centralized point, simultaneously place one or several copies of the software onto each of several remote target hosts (that is, "single point administration").

**Readme**

This keyword defines the "readme" attribute for the product object. A text file of the README information for the product; either the text value itself or a filename that contains the text.

**Realm**

The scope of the authority by which the Principal is authenticated in software security.

**Registers**

Or Registration. Determines which hosts are available for tasks (that is, which hosts have a daemon running). Also determines what depots are on a given host. Registration information consists of the depot or root's identifier (its path in the host file system). This information is maintained by the daemon which reads its own file at startup.

**Revision**

This keyword defines the "revision" attribute for the product object. The revision information (release number, version) for the product.

**Root Directory**

The directory on a target host in which all the files of the selected products will be installed. The default (/), can be changed to install into a directory that will eventually act as the root to another system. (See Alternate Root Directory.)

**Selection, Selection Phase**

The first phase of a software installation, copy, or remove operation, during which the user selects the software products to be installed on, copied to, or removed from the host.

**Servers**

A system on the network that acts as a software source for other nodes or clients on the network. Can be a network server or diskless server.

**Session Files**

Each invocation of an SD-UX software management command defines a session. The controller automatically saves invocation options, source information, software selections and host selections in special file locations before the operation actually starts.

**Shared Roots**

Directories located on a diskless server that are the location of operating system or application software that is shared among the system's diskless clients.

**Software Depots**

A system that contains one or more software products that can be installed on other systems. See Depot.

**Software Objects**

The term used to describe the objects that are packaged, distributed, installed and managed. The software *filesets*.

**Software Selection Window**

The second window to appear in the GUI. Helps you select the software or data files you want to install, copy or remove.

**Software Source**

The term used to reference a depot being used as the source of a `swinstall` or `swcopy` operation.

**Source**
See Software Source.

**Subproducts**
An optional grouping of filesets, used to partition a product which contains many filesets or to offer the user different views of the filesets.

**Systems**
Computers, either standalone or networked to other computers.

**swacl**
The SD-UX command that allows you to modify Access Control List permissions that provide software security.

**swagentd**
The daemon that provides various services, including initiation of communication between the controller and agent, and serving one or more depots simultaneously to multiple requesting agents on the host.

**swconfig**
The SD-UX command that configures software that has been installed, to make the software ready for use.

**swcluster**
The SD-UX command that launches `swinstall` and `swremove` to install or remove software from a HP-UX 10.0X NFS diskless cluster. It also launches the linkinstall and linkremove features to install or remove software from NFS client shared roots.

**swcopy**
The SD-UX command that copies software from any software source to a depot. The `swcopy` command can add products to an existing depot, replace products already on a depot and create a new depot.

**swgettools**
The SD-UX command that lets you install the new SW-DIST product from new SD media when you are updating the operating system. `swgettools` is necessary because the new media format may be different from the SD product that is already installed on the host system. This command is loaded from the new SD media using `cp`, `rcp` or `tar`.

**swinstall**
The SD-UX command that installs or updates software.

**swlist**
The SD-UX command that lists software objects, their attributes and their organization. It lists both installed software and software contained within a depot.

**swmodify**
The SD-UX command that allows you, from the command line, to change or add to the information in the IPD or depot catalog files.

**swpackage**
The SD-UX command used by a software supplier or system administrator to organize software products and package them into a depot. The depot can be accessed directly by the SD-UX software management tools, it can be mastered onto a CD-ROM depot, or it can be made available to other hosts by the `swagentd` command.

**swreg**
The command SD-UX uses to register depots.

**swremove**
The SD-UX command that removes software that is installed, or software contained with a (writable) depot.

**swverify**
The SD-UX command that verifies (for correctness and completeness) software that is installed, or software contained within a depot.

**Tag**
This keyword defines the *distribution.tag* or product "name" attribute for the destination depot (media) being created/modified by `swpackage`.

**Tape Media**
One type of software media. It uses `tar` to store software products and control files needed by the SD to use the media. It usually resides on a serial media such as a DDS, cartridge, nine-track, or other tape, though it can also be a regular file that contains the tar archive. Within the `tar` archive, directory and file entries are organized using the same structure as any other depot.

**Tape Depot**
The software in a tape depot is formatted as a `tar` archive. Encoded in this archive is the same SD media format directory hierarchy that is used in a directory depot. Tape depots such as cartridge tapes, DAT and 9-track tape are referred to by the file system path to the tape drive's device file.

**Tape Source**
A tape media being used as a source of software products. A tape source is identified by the name of the special file used to access the device where the media resides, or by the name of a regular file that contains the `tar` archive.

**Targets**
A directory on a host in which software is installed, copied or removed. They can operate on either a target root directory or a target depot.

**TUI**
A terminal user interface which works on an ASCII terminal and uses the keyboard to navigate.

**Title**
A one-line, full name attribute that further identifies the product.

**UUID**
This keyword defines the vendor's "uuid" attribute for the vendor object. Useful for NetLS vendors and for those who want to select products from two vendors who have chosen the same *vendor_tag*.

**Uname Attributes**
When a target is contacted for a software management operation, the system's four `uname` attributes - operating system name, release, version and hardware machine type - are obtained. Used to determine software compatibility with the proposed host.

**Unconfigure Scripts**
An optional, vendor-supplied script associated with a fileset that is executed by `swremove` before the removal of filesets begins.

**Updates**
Overwriting software objects already installed on the system and replacing them with new objects.

**Vendor**

If a vendor specification is included in the PSF, `swpackage` requires the "vendor" and "tag" keywords.

**Vendor_tag**

Associates the product or bundle with the last-defined vendor object, if that object has a matching *tag* attribute.

# Index